Main.c (main routine for EMCS-1 board - dsPIC30F6010A MCU)

```
//-------------------------------------------------------------------------//
//--BLDC Real-time Motor Control System (EMCS-1)---//
//--Author: Kostyantyn Koziy (SD0822 / NDSU)----------//
//--All rights reserved. Unauthorized copying prohibited.--//
//-------------------------------------------------------------------------//

// Configuration bits
//-----------------------------
_FOSC(CSW_FSCM_OFF & XT_PLL16);          // Internal Osc/ PLL x16
_FWDT(WDT_OFF);                          // WD off
//-----------------------------

// Include headers
//-----------------------------
#include "general.h"                     // Common definitions
#include "interrupts.h"                  // Interrupt routines
#include "main.h"                        // Main header
#include "DirectLEDs.h"                  // LEDs module
#include "max6959.h"                     // Buttons + LEDs module
#include "KS0108.h"                      // GLCD high-level routines
#include "GLCD_buffer.h"                 // GLCD buffer routines
#include "adc.h"                         // ADC routines
#include "pwm.h"                         // PWM routines
#include "p30f6010A.h"
//-----------------------------

// External variables
//-----------------------------
extern  unsigned int             SEGnum;              // MAX6959: 7Seg 4Dig num, 0000-9999
extern  packed_leds              SEGled;              // MAX6959: Dec point 0,1,2,3 + 4 leds
extern  packed_buttons           SEGbut;              // MAX6959: Buttons register
extern  packed_mess_types        Scr1Var[VAR_NUMBER]; // Display variables
extern unsigned long             adc_buffer[6];       // ADC buffer
extern unsigned long             EstimatedSpeedOut;   // Speed from ADC ISR (not averaged)
extern unsigned long             TempDisplayAvrSpeed; // Used to display speed on GLCD with 0.2s update
//-----------------------------


// Global variables
//-----------------------------
                                 // BLDC coil switch table
const unsigned int StateLoTable[8] =
{
  0b0000000000111111,            // 000000
  0b0010000000011110,            // P11110
  0b0010000000011011,            // P11011
  0b0000001000111001,            // 1110P1
  0b0000001000101101,            // 1011P1
  0b0000100000100111,            // 10P111
  0b0000100000110110,            // 11P110
  0b0000000000111111             // 000000
};
flags_packed Flags;              // Control flags
//-----------------------------

// Local variables
//-----------------------------
                                 // Speed variables:
unsigned int        SpeedBuffer100ms[16];  // To average speed
float               SpeedAverage1600ms;    // Averaged speed
unsigned int        SetSpeedToPWMi;        // Conversion from speed to PWM
unsigned int        CurrentMesSpeed;       // Current momentary speed
unsigned int        SetSpeed;              // Reference speed (unsigned)

                                 // PID variables:
unsigned    int     Vmeas_scaled;          // Measured speed scaled to 0..511
signed      long    Increment;             // PID output -> PWM
signed      int     SpeedError;            // Current speed error
signed      long    SpeedProp;             // Proportional element
signed      long    SpeedInt;              // Integral element
signed      long    SpeedDer;              // Derivative element
unsigned    int     Vref_scaled;           // Referance speed scaled to 0..511
signed      int     LastSpeedError;        // Previous speed error
unsigned    int     Duty;                  // Final calculated duty cycle

                                 // Menu variables:
unsigned char menu_pos;                    // Current menu element num
//-----------------------------
```

```
// Main routine
//------------------------------
int main(void)
{
                                        // Initialization procedures:
  InitDirectLEDs();                     // Initialize LED and button ports
  GLCD_Initialize();                    // Initialize GLCD ports and ks0108
  GLCD_ClearScreen();                   // Initialize GLCD Screen
  GLCD_Screen1_Var_Ini();               // Initialize display variables
  InitI2C();                            // Initialize I2C bus
  InitMax6959();                        // Initialize IO chip
  Init_Timer1();                        // Initialize Timer1 for multitasking
  Init_ADC();                           // Initialize ADC for coil switching and monitoring
  Init_PWM();                           // Initialize PWM generator
  PGAx10                                // Set PGA on 10V/V = 1V/A sens

                                        //  Initialization regs:
  INTCON1bits.NSTDIS = 0;               // Enable nested interrupts

                                        // Temporal test variables:
  signed long i;
  unsigned int loop;

                                        // Start LED and System Status set:
  LED_LD1G = 1;                         // No Fault - Green
  PORTW
  LED_LD1R = 0;
  PORTW
  LED_LD3R = 0;                         // Out MENU - Green
  PORTW
  LED_LD3G = 1;
  PORTW
  LED_D11  = 1;                         // No Run, In Pause
  LED_D12  = 0;

  SS_Id                                 // Idle msg
  df_PP = 1;                            // Pause in Scr

// MAIN infinite loop
while(1) // WHILE 1
{
  if (BUT_SW3)                          // Get inside main menu...
  {
        SS_Mn                           // In Menu msg

        LED_LD3R=1;                     // In MENU - Red
        PORTW
        LED_LD3G=0;
        PORTW

        LED_D11=1;                      // No STOP, In Pause
        LED_D12=0;
        da_S = 0;
        SEGbut.buttons=0;
        menu_pos = 1;

        while (1) // WHILE 2
        {
              if (BUT_SW8)              // Enter is PRESSED ?
              {
          PORTW
                                        LED_LD3G = 1;
        PORTW
                                        LED_LD3R = 1;           // In MENU, par - Red + Green

                                        LED_D11  = 1;           // No STOP, In Pause
                                        LED_D12  = 0;
                                        switch (menu_pos)       // ...set appropriate element...
                                        {
                                              case 1:           // Ref speed...
                                                    da_Vref = 2; SEGbut.buttons=0;
                                                    SS_RV   // Ref Speed msg
                                                    while (1)
                                                    {
                                                          i = (((((signed long)POT_R52*100)/1023)-50)*100);
                                                          if ((i < 800) && (i > -800))
                                                          {
                                                                dn_Vref = 0;
                                                          } // IF
                                                          else
```

```
                                {
                                        dn_Vref = i + ((((signed long)POT_R59*100)/1023)-50);
                                } // ELSE
                                if (dn_Vref >= 0) SEGnum = (unsigned int) dn_Vref;
                                else        SEGnum = (unsigned int)(-dn_Vref);
                                db_Vref = POT_R52;
                                if (BUT_SW8) {SS_Mn; SEGnum = 0; break;}        // Get away from current menu element...
                                } // WHILE

                        da_Vref = 4; SEGbut.buttons=0;
                break;

                case 2:
                                // Feedback flag...
                        da_Fb = 2; SEGbut.buttons=0;
                        SS_Fb     // Flag msg
                        while (1)
                        {               SEGnum = dn_Fb;
                                if (BUT_SW6  || BUT_SW9) dn_Fb = 0;              // Clear flag - no speed feedback, open loop
                                if (BUT_SW10 || BUT_SW7) dn_Fb = 1;              // Set flag - speed feedback, close loop (PID)
                                if (BUT_SW8) {SS_Mn; SEGnum = 0; break;}        // Get away from current menu element...
                        } // WHILE
                                da_Fb = 4; SEGbut.buttons=0;
                break;

                case 3:                 // Set Derivative coefficient (*2^3)...
                        da_D = 2; SEGbut.buttons=0;
                         SS_Dc    // Kd msg
                        LED_DP3 = 1;
                        while (1)
                        {
                                SEGnum = dn_D;
                                dn_D = (((POT_R52*100)/1024)/10)*100 + ((POT_R59*100)/1024);
                                if (BUT_SW8) {SS_Mn; SEGnum = 0; break;}        // Get away from current menu element...
                        } // WHILE
                        LED_DP3 = 0;
                        da_D = 4; SEGbut.buttons=0;
                break;

                case 4:                 // Set Integral coefficient (*2^3)...
                        da_I = 2; SEGbut.buttons=0;
                        SS_Ic      // Ki msg
                        LED_DP3 = 1;
                        while (1)
                        {
                                SEGnum = dn_I;
                                dn_I = (((POT_R52*100)/1024)/10)*100 + ((POT_R59*100)/1024);
                                if (BUT_SW8) {SS_Mn; SEGnum = 0; break;}        // Get away from current menu element...
                        } // WHILE
                        LED_DP3 = 0;
                        da_I = 4; SEGbut.buttons=0;
                break;

                case 5:                 // Set Proportional coefficient (*2^3)...
                        da_P = 2; SEGbut.buttons=0;
                        SS_Pc     // Kp msg
                        LED_DP2 = 1;
                        while (1)
                        {
                                SEGnum = dn_P;
                                dn_P = ((((POT_R52*100)/1024)/10)*10) + ((POT_R59*100)/1024)/10;
                                if (BUT_SW8){SS_Mn; SEGnum = 0; break;}        // Get away from current menu element...
                        } // WHILE
                        LED_DP2 = 0;
                        da_P = 4; SEGbut.buttons=0;
                break;

                default:
                break;
        } // SWITCH (menu_pos)
                                        // Reset status and LED state...
        LED_LD3R = 1;                   // In MENU - Red
        PORTW
        LED_LD3G = 0;
        PORTW
        LED_D11  = 1;                   // No STOP, In Pause
        LED_D12  = 0;

} // IF (BUT_SW8)
```

```
if (BUT_SW10 || BUT_SW7)                    // Increase menu num element...
{
   menu_pos++; if (menu_pos > 5) menu_pos = 5;
   SEGbut.buttons=0;
} // IF

if (BUT_SW6 || BUT_SW9)                     // Decrease menu num element...
{
   menu_pos--; if (menu_pos < 1) menu_pos = 1;
   SEGbut.buttons=0;
} // IF

switch (menu_pos)                           // Highlight corresponded menu element...
{
   case 1:
            da_Vref = 4; da_Fb = 1;
   break;

   case 2:
            da_Vref = 1; da_Fb = 4; da_D = 1;
   break;

   case 3:
            da_Fb = 1; da_D = 4; da_I = 1;
   break;

   case 4:
            da_D = 1; da_I = 4; da_P = 1;
   break;

   case 5:
            da_P = 4; da_I = 1;
   break;

   default:
   break;
} // SWITCH

if (BUT_SW3)                        // Get away from main menu to Idle...
  {
   LED_LD3G = 1;
   PORTW
   LED_LD3R = 0;                    // Out MENU - Green
   PORTW
   LED_D11  = 1;                    // No Run, In Pause
            LED_D12  = 0;

   da_Vref = 1; da_Fb = 1; da_D = 1; da_I = 1; da_P = 1;
   da_PP = 1;
   da_S = 4;
             SEGbut.buttons=0;
   SS_Id
   df_PP = 1;
   break;
  } // IF

 } // WHILE 2
} // IF

if (BUT_SW5)                                // Start motor command...
{
            LED_LD3R = 0;                   // In Run
   PORTW
   LED_LD3G = 0;
   PORTW
   LED_D12 = 1;                     // Start Led
   LED_D11 = 0;

   switch (dn_Fb)                   // Open loop/close loop switch
   {
            case 0:
            //-----------------------------------------------------------------Open Loop code
                     da_PP = 4;
                     df_PP = 0;
                     df_S = 1;
                     da_S = 1;
                     SEGbut.buttons=0;
                     SS_Ro      // Open loop msg
```

```
while (1) // WHILE 3
{
  if (Flags.SpeedReady)                         // If new speed is ready from ADC ISR...
  {
        SpeedAverage1600ms = 0;                 // Calculate average speed
        for (loop = 0; loop <15; loop++)
        {
              SpeedBuffer100ms[loop+1] = SpeedBuffer100ms[loop];
              SpeedAverage1600ms += SpeedBuffer100ms[loop+1];
        }
        SpeedBuffer100ms[0] = EstimatedSpeedOut;
        SpeedAverage1600ms += SpeedBuffer100ms[0];
        SpeedAverage1600ms = SpeedAverage1600ms/16;
        TempDisplayAvrSpeed = SpeedAverage1600ms;

                                                // If measured speed is too small - blink
        if ((TempDisplayAvrSpeed <= 1200) && (Flags.RotDirection))
        {
              da_Vm = 3; ab_Vm = 3;
        }
        else
        {
              da_Vm = 1; ab_Vm = 1;
        }

                                                // Get new reference speed from potentiometer...
        i = (((((signed long)POT_R52*100)/1023)-50)*2);
        if ((i < 25) && (i > -25))              // For <25% speed = set to 0, can't control
        {
              dn_Vref = 0; db_Vref = 512;
        }
        else
        {
              dn_Vref = i; db_Vref = POT_R52;
        }

        if (dn_Vref > 0)                        // Set correct direction...
        {
              SetSpeed = (unsigned int)dn_Vref; Flags.RotDirection = 1;    LED_D14 = 1; PORTW LED_D15 = 0;
        }
        else if (dn_Vref < 0)
        {
              SetSpeed = (unsigned int)(-dn_Vref); Flags.RotDirection = 2; LED_D14 = 0; PORTW LED_D15 = 1;
        }                                       // Brake the motor when changing direction...

      else if ((dn_Vref == 0) && (SpeedAverage1600ms > 1000))
        {
              SEGnum = 0;
              Flags.BrakingDone = 0;
              Flags.RotDirection = 0;
              while (!Flags.BrakingDone)    // Wait until speed will come down...
              {
                    LED_D14 = 1; PORTW LED_D15 = 1;             // Continue to get speed..
                    i = (((((signed long)POT_R52*100)/1023)-50)*2);
                    dn_Vref = i; db_Vref = POT_R52;
              } // WHILE
        } // ELSE IF

                                                // Set PWM from 25...100% (3000...290)
        SetSpeedToPWMi = ((125 - SetSpeed)*((PWMmin - PWMmax)/100)) + PWMmax;
        if (Flags.RotDirection)                                 // Display speed on LED
        {
              SEGnum = SpeedAverage1600ms;
        } // IF
        else {SEGnum = 0;}

                                                // Set PWM registers...
        PDC1 = SetSpeedToPWMi;                  // PWM 1, 2 and 3
        PDC2 = PDC1;
        PDC3 = PDC1;
        FIRE_ENA = 0;                           // Enable PWM driver output...
        Flags.SpeedReady = 0;                   // Reset Speed ready flag
  } // IF (Flags.SpeedReady)
```

```
if (BUT_SW4)                            // Stop motor
  {
   LED_LD3R = 0;                        // Out Menu
   PORTW
   LED_LD3G = 1;
   PORTW
   LED_D14 = 0;                         // No direction
   PORTW
   LED_D15 = 0;

   LED_D12 = 0;                         // Stop Led
   LED_D11 = 1;
            SS_Id                       // Idle msg

   da_PP = 1;
   df_PP = 1;
   df_S = 0;
   da_S = 4;
   SEGbut.buttons=0;
   da_Vm = 1; ab_Vm = 1;
   Flags.RotDirection = 0;
   TempDisplayAvrSpeed = 0;
   adc_buffer[1] = 0;
   SEGnum = 0;
   break;
  } // IF (BUT_SW4)

 } // WHILE 3

break; // case0

case 1:
//-----------------------------------------------------------------Closed Loop (PID) code
   da_PP = 4;
   df_PP = 0;
   df_S = 1;
   da_S = 1;
   SEGbut.buttons=0;
   LED_D12 = 1;
   LED_D11 = 0;
   SS_Rc
   FIRE_ENA = 1;                        // Disable PWM driver output…

   if (dn_Vref > 1000)                  // Set appropriate direction flag...
   {
            SetSpeed = dn_Vref; Flags.RotDirection = 1; LED_D14 = 1; PORTW LED_D15 = 0;
   }
   else if (dn_Vref < -1000)
   {
            SetSpeed = (-dn_Vref); Flags.RotDirection = 2; LED_D14 = 0; PORTW LED_D15 = 1;
   }
   else                                                         // Speed is too low or 0.
   {
            dn_Vref = 0; Flags.RotDirection = 0; LED_D14 = 0; PORTW LED_D15 = 0;
   }
                                        // Scale reference speed to 0..511
   Vref_scaled = ((unsigned long)SetSpeed * PWMres)/(Vmax - Vmin);
   while (1) // WHILE 4
   {
            // PID loop
            if (Flags.SpeedReady)       // When new speed from ADC ISR is ready...
            {                                                           // Scale it...
                     Vmeas_scaled = ((unsigned long)(EstimatedSpeedOut*(PWMres))/(Vmax-Vmin));
                     Increment = 0;
                                        // Estimate current speed error...
                     SpeedError = (Vref_scaled - Vmeas_scaled);

                                        // Proportional term...
                     SpeedProp = (((dn_P<<PIDcoefsh)*SpeedError)/10);                   // Kp = (00.0 .. 99.9) * 16

           // Integral term...
                     SpeedInt =         (((dn_I<<PIDcoefsh)*(SpeedInt + SpeedError))/100);      // Ki = (0.00 .. 9.99) * 16
                     if (SpeedInt > PWMres) SpeedInt = PWMres;
                     if (SpeedInt < (-(PWMres))) SpeedInt = -(PWMres);

           // Derivative term...
                     SpeedDer =  (((dn_D<<PIDcoefsh)*(SpeedError - LastSpeedError))/100);    // Kd = (0.00 .. 9.99) * 16
                     if (SpeedDer > PWMres) SpeedDer = 511;
                     if (SpeedDer < (-(PWMres))) SpeedDer = -511;
```

```
            LastSpeedError = SpeedError;
                        // New speed...
            Increment = SpeedProp + SpeedInt + SpeedDer;

                        // Scale back new speed...
            Increment = Increment>>PIDcoefsh;

          // Check margins...
            if (Increment >= (signed int)PWMres)    Duty = PWMmin;
            else if (Increment <= (signed int)0) Duty = PWMmax;

                        // ... and estimate new PWM
            else Duty = (unsigned int)(PWMmin - ((unsigned long)(Increment*(PWMmin-PWMmax))/PWMres));

                        // Display current speed if rotating...
            if (Flags.RotDirection) {SEGnum = SpeedAverage1600ms;}
            else {SEGnum = 0;}

                        // Set PWM registers...
            PDC1 = Duty;                            // PWM 1, 2 and 3
            PDC2 = PDC1;
            PDC3 = PDC1;
            FIRE_ENA = 0;                           // Enable PWM output
            SpeedAverage1600ms = 0;    // Estimate average speed to display
            for (loop = 0; loop <15; loop++)
            {
                        SpeedBuffer100ms[loop+1] = SpeedBuffer100ms[loop];
                        SpeedAverage1600ms += SpeedBuffer100ms[loop+1];
            }
                        SpeedBuffer100ms[0] = EstimatedSpeedOut;
                        SpeedAverage1600ms += SpeedBuffer100ms[0];
                        SpeedAverage1600ms = SpeedAverage1600ms/16;
                        TempDisplayAvrSpeed = SpeedAverage1600ms;

                        Flags.SpeedReady = 0;       // Reset ready flag
            } // IF (Flags.SpeedReady)



            if (BUT_SW4)                            // Stop motor...
            {
                        LED_D14 = 0;
                        da_PP = 1;
                        df_PP = 1;
                        df_S = 0;
                        da_S = 4;
                        SEGbut.buttons=0;

                        LED_LD3R = 0;               // Out Menu
                        PORTW
                        LED_LD3G = 1;
                        PORTW
                        LED_D14 = 0;                // No direction
                        PORTW
                        LED_D15 = 0;

                        LED_D12 = 0;                // Stop Led
                        LED_D11 = 1;
                        SS_Id                       // Idle msg

                        adc_buffer[1] = 0;
                        Flags.RotDirection = 0; // Brake down...
                        da_Vm = 1; ab_Vm = 1;
                        TempDisplayAvrSpeed = 0;
                        SEGnum = 0;

                        break;
            }
   } // WHILE 4
break;

default:
break;
       } // SWITCH (dn_Fb)
     } // IF (BUT_SW5)
   } // WHILE 1
} // main
```

Adc.h

```
// Functions
//------------------------------
void Init_ADC (void);                          // Initiaize ADC
void _ISR _ADCInterrupt(void); // ADC ISR
//------------------------------
```

Adc.c

```
// Include headers
//------------------------------
#include "p30f6010A.h"
#include "general.h"
#include "DirectLEDs.h"
#include "string.h"
//------------------------------

// External variables
//------------------------------
extern  flags_packed              Flags;     // Control flags
extern  unsigned int   StateLoTable[8];    // BLDC coil switching table
//------------------------------

// Global variables
//------------------------------
unsigned long                     adc_buffer[6];          // ADC register buffer
unsigned int                      MonitorBufferCount;     // Monitor (temperature/voltage) counter (for averaging)
unsigned int                      CurrentBufferCount;     // Monitor (current) counter (for averaging)
unsigned long                     EstimatedSpeedOut;      // Estimated speed
unsigned char                     speed_avr1[4];          // Window of Encoder position values (to lock reset)
//------------------------------

// Local variables
//------------------------------
unsigned int                      CountTimerOuts;         // Speed values counter (to estimate speed)
unsigned int                      TotalTimerSumm;         // Speed total value (summ - to estimate speed)
unsigned int                      Count1094;              // Counter - 0.1s
unsigned int                      Count400;               // Counter - max between reset points
unsigned char                     seq_switch = 1;         // Rotation direction state machine
unsigned char                     coil_num;               // Current coil switch table number
//------------------------------

// Init_ADC routine
void Init_ADC (void)
{                                 // Initialize Analog pins:
TRISBbits.TRISB8 = 1;  ADPCFGbits.PCFG8     = 0;     // AN8 - analog input - R52
TRISBbits.TRISB9 = 1;  ADPCFGbits.PCFG9     = 0;     // AN9 - analog input - R59
TRISBbits.TRISB10 = 1; ADPCFGbits.PCFG10    = 0;     // AN10 - analog input - Encoder
TRISBbits.TRISB11 = 1; ADPCFGbits.PCFG11    = 0;     // AN11 - analog input - Vbus
TRISBbits.TRISB6 = 1;  ADPCFGbits.PCFG6     = 0;     // AN6 - analog input - Module temperature
TRISBbits.TRISB2 = 1;  ADPCFGbits.PCFG2     = 0;     // AN2 - analog input - Phase current

ADCON2bits.VCFG = 0b111;                  // Voltage Ref to AVss/Avdd
ADCON2bits.CHPS   = 0;                    // Sample channel CH0
ADCHSbits.CH0SA   = 0;                    // Mux A = 0
ADCHSbits.CH0NA   = 0;                    // Mux A is referenced to VREF-
ADCHSbits.CH0SB = 0;                      // Mux B = 0
ADCHSbits.CH0NB = 0;                      // Mux B is referenced to VREF-
ADCON2bits.ALTS   = 0;                    // Alternate sampling disabled
ADCON2bits.BUFM = 0;                      // Single 16-words buffer
ADCON2bits.SMPI = 5;                      // Interrupt on 6th sample
ADCON2bits.CSCNA= 1;                      // Scan CH0+ inputs
ADCSSL = 0b0000111101000100;              // Scan AN 2,6,8,9,10,11

ADCON1bits.FORM = 0b00;                   // Integer format

ADCON3bits.ADCS = 63;                     // Fisr = (33.3ns(ADCS+1)/2)*6ch = 15.232us*6ch=10941.88Hz/ISR = 10094Hz
ADCON1bits.SSRC = 0b111;                  // Internal counter ends sampling and starts conversion
ADCON1bits.ASAM = 1;                      // Start auto sampling

IFS0bits.ADIF = 0;                        // Clear interupt flag
IPC2bits.ADIP = 5;                        // Set ISR Priority to 5 (Over Timer1)
IEC0bits.ADIE = 1;                        // Enable interrupt
ADCON1bits.ADON = 1;                      // Turn ADC ON
} //Init_ADC
```

```
// _ADCInterrupt routine
void __attribute__((interrupt, no_auto_psv)) _ADCInterrupt(void)
{
//--------------------------     Back Up ADC buffers...
adc_buffer[4] = ADCBUF4;                    // Rotor position (0..511)
adc_buffer[2] = ADCBUF2;                    // Potentiometer R52
adc_buffer[3] = ADCBUF3;                    // Potentiometer R59

//--------------------------     Measure Phase C current, one direction, add to average buffer...
if ((coil_num == 1) && (CurrentBufferCount < 1024))
{
   adc_buffer[1] += ADCBUF0;
   CurrentBufferCount++;          // Inc average buffer count
}

//--------------------------     Estimate rotot position, commutate coils...
switch (Flags.RotDirection)
{
   case 0b00:                                // Braking.../Idle
            if (EstimatedSpeedOut > 1000)
            {
                     PDC1 = 300;          // PWM 1, 2 and 3
                     PDC2 = 300;
                     PDC3 = 300;
                     FIRE_ENA = 0;
                     coil_num = 1;
                     Flags.BrakingDone = 0;
            }
            else                              // Braking finished...
            {
                     FIRE_ENA = 1;
                     coil_num = 0;
                     Flags.BrakingDone = 1;
            }
   break;
   case 0b01:                                        // Forward rotation
            coil_num = (((512-adc_buffer[4]+ADVANCED_SWITCHING_FW)/14)%6)+1;
   break;
   case 0b10:                                        // Backward rotation
            coil_num = 7-(((((adc_buffer[4]+ADVANCED_SWITCHING_RW))/14))%6)-1;
   break;
   default:                                          // Error case, must never happen
            coil_num = 0; Flags.FLT = 0b001;         // Incorrect Motor Op set
   break;
}
 OVDCON = StateLoTable[coil_num];                    // Update PWM output table

//--------------------------               Estimate Temperature and Bus Voltage, add to average buffer...
if ((MonitorBufferCount < 256))
{
   adc_buffer[0] += ADCBUF1;                         // Temperature
   adc_buffer[5] += ADCBUF5;                         // Bus Voltage
   MonitorBufferCount++;                             // Inc average buffer count
}

//--------------------------               Speed estimator
                                                     // Find window of similar values in Encoder analog output...
speed_avr1[3] = speed_avr1[2];
speed_avr1[2] = speed_avr1[1];
speed_avr1[1] = speed_avr1[0];
speed_avr1[0] = (ADCBUF4>>5)+1;          // Make sure that all values 0..3 are the same...
int avr_timer = (speed_avr1[0] + speed_avr1[1] + speed_avr1[2] + speed_avr1[3])>>2;

Count400++;                                          // Increase between-resets (BRC) counter (400*(1/10941) max time, 50*(1/10941) min time)
                                                     // If over 400 - reset BRC, and flag that reset was done because of overflow (no rotation)
if (Count400 == 401) {Count400 = 0; Flags.RPS = 0;}

if (!((avr_timer+5)%8))                              // If current rotor position = reset position... (2 poles)
{
   if ((Flags.RPS) && (Count400 > 60))     // ...check if previous BRC reset was done because of rotor reset position...
   {
                     TotalTimerSumm += Count400;          // Add BR counter value to average buffer
                     CountTimerOuts++;                    // Increase average buffer count
   }
   Flags.RPS = 1;                                    // Set RPS (reset was done because of reset position)
   Count400 = 0;                                     // Reset BR counter - ready for count till next reset position
}

Count1094++;                                         // Increase 0.1s speed output counter
```

```
                                        // If 0.1s passed and at least 1 speed value available...
if ((Count1094 == 1094) && (CountTimerOuts != 0))
{                                       // ... estimate speed (60/(Poles*ADC_sample_period))/(Average BRC counter values)
   EstimatedSpeedOut = 328256/(TotalTimerSumm/CountTimerOuts);
   TotalTimerSumm = 0;                  // Clear variables...
   CountTimerOuts = 0;
   Count1094 = 0;
   Flags.SpeedReady = 1;                // Indicate that new speed is available
}
else if ((Count1094 == 1094))                   // If 0.1s passed and NO speed values available...
{
   EstimatedSpeedOut = 800;             // Means no rotation or speed lower then 800RPM
   TotalTimerSumm = 0;
   CountTimerOuts = 0;
   Count1094 = 0;
   Flags.SpeedReady = 1;
   Flags.RPS = 0;
}
//--------------------------

IFS0bits.ADIF = 0;                              // Clear interrupt flag
} // _ADCInterrupt
```

Control.h

```
// Functions
//------------------------------
float SpeedFilter(float);              // Filter routine
                                       // invar - next sample
//------------------------------
```

Control.c

```
// Include headers
//------------------------------
#include "control.h"
#include "p30f6010A.h"
#include "general.h"
//------------------------------

// Functions
//------------------------------

// 3rd Order Low Pass Butterworth for speed sequence
// Sample Frequency = 10.00 Hz
// With 1st Order Sin(X)/X Correction
// Pass Band Frequency = 1.000 Hz
// SpeedFilter routine
float SpeedFilter(invar)
float invar;
{
    float sumnum, sumden;
   int i;
    static float delay[5] = {0.0,0.0,0.0,0.0,0.0};
    static float znum[5] = {
        0.0,
        0.0,
        0.0,
        0.0,
        .1535
    };
    static float zden[4] = {
        -4.554e-02,
        -9.257e-02,
        .9149,
        -1.623
    };
        sumden=0.0;
        sumnum=0.0;
        for (i=0;i<=3;i++){
            delay[i] = delay[i+1];
            sumden += delay[i]*zden[i];
            sumnum += delay[i]*znum[i];
        }
        delay[4] = invar-sumden;
        sumnum += delay[4]*znum[4];
        return sumnum;
} // SpeedFilter

//------------------------------
```

DirectLEDs.h

```
// Definitions
//------------------------------
        // LEDs:
#define LED_D15         PORTDbits.RD13
#define LED_D14         PORTAbits.RA9
#define LED_LD1G        PORTDbits.RD14
#define LED_LD1R        PORTDbits.RD15
#define LED_LD3R        PORTEbits.RE6
#define LED_LD3G        PORTEbits.RE7
        // Buttons:
#define BUT_SW1         PORTAbits.RA14  // Active Low
#define BUT_SW2         PORTGbits.RG9     // Active Low

        // PGA control
#define PGA_G0          PORTFbits.RF4
#define PGA_G1          PORTFbits.RF5
#define PGA_G2          PORTFbits.RF8
#define PGAx10          PGA_G0=0;PGA_G1=0;PGA_G2=1;
#define PGAx50          PGA_G0=0;PGA_G1=1;PGA_G2=1;

#define PORTW           asm("nop");asm("nop");

//------------------------------

// Functions
//------------------------------
void InitDirectLEDs(void);      // Initialize direct LEDs and buttons
//------------------------------
```

DirectLEDs.c

```
// Include headers
//------------------------------
#include "general.h"
#include "p30f6010A.h"
#include "DirectLEDs.h"
//------------------------------

// Functions
//------------------------------

// Initialize direct LEDs and buttons
void InitDirectLEDs (void) {
        // Direct LEDs:
TRISDbits.TRISD13 = 0;      // LED D15

ADPCFGbits.PCFG9 = 1;       // RA9 as digital for D15
TRISAbits.TRISA9 = 0;       // LED D14

TRISDbits.TRISD14 = 0;      // LED LD1G
TRISDbits.TRISD15 = 0;      // LED LD1R
TRISEbits.TRISE6 = 0;       // LED LD3R
TRISEbits.TRISE7 = 0;       // LED LD3G

ADPCFGbits.PCFG9 = 1;       // RA9 as digital for D15
TRISAbits.TRISA9 = 0;       // LED D14

        // Direct Buttons:
ADPCFGbits.PCFG14 = 1;      // RA9 as digital for SW1
TRISAbits.TRISA14 = 1;      // Button SW1

TRISGbits.TRISG9  = 1;      // Button SW2

        // PGA control:
TRISFbits.TRISF4 = 0;       // G0
TRISFbits.TRISF5 = 0;       // G1
TRISFbits.TRISF8 = 0;       // G2
}; // InitDirectLEDs
//------------------------------
```

GLCD.h

```
// Definitions
//------------------------------
#define LCD_RST                     PORTFbits.RF6
#define LCD_EN                      PORTFbits.RF7
#define LCD_RS                      PORTCbits.RC1
#define LCD_RW                      PORTCbits.RC3
#define LCD_CS1                     PORTGbits.RG0
#define LCD_CS2                     PORTGbits.RG1

#define LCD_DAT                     PORTD

#define DISPLAY_STATUS_BUSY         0x80
//------------------------------

// Functions
//------------------------------
void GLCD_Delay(void);                                          // One Tcy delay
void GLCD_InitializePorts(void);                                // Initialize dsPIC ports to work with GLCD
void GLCD_EnableController(unsigned char controller);           // Enable controller CS1/CS2
void GLCD_DisableController(unsigned char controller);          // Disable controller CS1/CS2
unsigned char GLCD_ReadStatus(unsigned char controller);        // Read CS1/CS2 controller Status
                                                                // controller - number

void GLCD_WriteCommand(unsigned char commandToWrite,            // Write command to CS1/CS2 controller
                       unsigned char controller);               // commandToWrite - command, controller - number

unsigned char GLCD_ReadData(void);                              // Read data from CS1/CS2 controller

void GLCD_WriteData(unsigned char dataToWrite);                 // Write data to current position
                                                                // dataToWrite - data

unsigned char GLCD_ReadByteFromROMMemory(char * ptr);           // Read byte from ROM
                                                                // * ptr - pointer to data location
//------------------------------
```

GLCD.c

```
// Include headers
//------------------------------
#include "general.h"
#include "p30f6010A.h"
#include "GLCD.h"
//------------------------------

// External variables and functions
//-----------------------------------
extern unsigned char screen_x, screen_y;        // Pixel location on GLCD
extern unsigned char GLCD_state_machine;        // Sequential counter for output to GLCD function
extern unsigned char GLCDv_state_machine;       // Sequential counter for graphic gnerator function
extern unsigned int  mem_count;                 // Output symbols to RAM only at the beginning of output to GLCD
extern void GLCD_WriteVariables(void);          // Sequentional graphic generator of Screen Variables
//-----------------------------------

// Functions
//-----------------------------------

// One Tcy delay
void GLCD_Delay(void)
{
   asm("nop");
}

// Initialize dsPIC ports to work with GLCD
void GLCD_InitializePorts(void)
{
   TRISD &= 0b1111111100000000;       // LCD data line as output;

   TRISGbits.TRISG0 = 0;              // CS1/RG0 - output
   TRISGbits.TRISG1 = 0;              // CS2/RG1 - output
   TRISFbits.TRISF6 = 0;              // RST/RF6 - output (Reset)
   TRISCbits.TRISC3 = 0;              //  RW/RC3 - output
   TRISFbits.TRISF7 = 0;              //  EN/RF7 - output (Enable)
   TRISCbits.TRISC1 = 0;              //  DI/RC1 - output (Register Select)

   int i;
   LCD_RST = 0;
   for (i = 0; i<1000; i++);
```

```
   LCD_RST = 1;
   GLCD_state_machine = 0;                  // Reset GLCD and Write variable machines
   GLCDv_state_machine = 0;
} // GLCD_InitializePorts

// Enable controller CS1/CS2
void GLCD_EnableController(unsigned char controller)
{
   switch(controller)
   {
                case 0 : LCD_CS1 = 1;
                break;

                case 1 : LCD_CS2 = 1;
                break;
   }
} // GLCD_EnableController

// Disable controller CS1/CS2
void GLCD_DisableController(unsigned char controller)
{
   switch(controller)
   {
                case 0 : LCD_CS1 = 0;
                break;

                case 1 : LCD_CS2 = 0;
                break;
   }
} // GLCD_DisableController

// Read CS1/CS2 controller Status
// controller - number
unsigned char GLCD_ReadStatus(unsigned char controller)
{
   unsigned char status;
   TRISD |= 0b0000000011111111;          // Set port D to input
   LCD_RW = 1;
                asm("nop");
                asm("nop");
                asm("nop");
   LCD_RS = 0;
   switch(controller)
   {
                case 0 : LCD_CS1 = 1;
                break;

                case 1 : LCD_CS2 = 1;
                break;
   }
   LCD_EN = 1;
                asm("nop");
                asm("nop");
                asm("nop");
   status = LCD_DAT;
   LCD_EN = 0;
   switch(controller)
   {
                case 0 : LCD_CS1 = 0;
                break;
                case 1 : LCD_CS2 = 0;
                break;
   }
   return (status);
} // GLCD_ReadStatus

// Write command to CS1/CS2 controller
// commandToWrite - command, controller - number
void GLCD_WriteCommand(unsigned char commandToWrite, unsigned char controller)
{
   while(GLCD_ReadStatus(controller)&DISPLAY_STATUS_BUSY)
   {
                if (mem_count < VAR_NUMBER)          GLCD_WriteVariables();
                // Generate VAR_NUM Variables in Video RAM during first GLCD bytes...
   }

   TRISD &= 0b1111111100000000;          // LCD data line as output;
   LCD_RW = 0;
                asm("nop");
                asm("nop");
```

```
		asm("nop");
	LCD_RS = 0;
	switch(controller)
	{
		case 0 : LCD_CS1 = 1;
		break;

		case 1 : LCD_CS2 = 1;
		break;
	}

	unsigned int temp_port = LCD_DAT;
	temp_port &= 0b1111111100000000;
	temp_port += commandToWrite;
	PORTD = temp_port;			// Output data

	LCD_EN = 1;
		asm("nop");
		asm("nop");
		asm("nop");
	LCD_EN = 0;
	switch(controller)
	{
		case 0 : LCD_CS1 = 0;
		break;

		case 1 : LCD_CS2 = 0;
		break;
	}
} // GLCD_WriteCommand

// Read data from CS1/CS2 controller
unsigned char GLCD_ReadData(void)
{
	unsigned char data;

	while(GLCD_ReadStatus(screen_x / 64)&DISPLAY_STATUS_BUSY)
	{
	//		if (mem_count < VAR_NUMBER)		GLCD_WriteVariables();
	}

	LCD_RW = 1;
		asm("nop");
		asm("nop");
		asm("nop");
	LCD_RS = 1;
	switch(screen_x / 64)
	{
		case 0 : LCD_CS1 = 1;
		break;

		case 1 : LCD_CS2 = 1;
		break;
	}
	LCD_EN = 1;
		asm("nop");
		asm("nop");
		asm("nop");
	data = LCD_DAT;
	LCD_EN = 0;
	switch((screen_x / 64))
	{
		case 0 : LCD_CS1 = 0;
		break;

		case 1 : LCD_CS2 = 0;
		break;
	}
	screen_x++;
	return (data);
} // GLCD_ReadData

// Write data to current position
// dataToWrite - data
void GLCD_WriteData(unsigned char dataToWrite)
{
	while(GLCD_ReadStatus(screen_x / 64)&DISPLAY_STATUS_BUSY)
	{
		if (mem_count < VAR_NUMBER)		GLCD_WriteVariables();
	}
```

```
    TRISD &= 0b1111111100000000;        // LCD data line as output;
    LCD_RW = 0;
                asm("nop");
                asm("nop");
                asm("nop");
    LCD_RS = 1;

    unsigned int temp_port = LCD_DAT;
    temp_port &= 0b1111111100000000;
    temp_port += dataToWrite;
    LCD_DAT = temp_port;                // Output data

    switch(screen_x / 64)
    {
                case 0 : LCD_CS1 = 1;
                break;

                case 1 : LCD_CS2 = 1;
                break;
    }
    LCD_EN = 1;
                asm("nop");
                asm("nop");
                asm("nop");
    LCD_EN = 0;
    switch((screen_x / 64))
    {
                case 0 : LCD_CS1 = 0;
                break;

                case 1 : LCD_CS2 = 0;
                break;
    }
    screen_x++;
} // GLCD_WriteData

// Read byte from ROM
// * ptr - pointer to data location
unsigned char GLCD_ReadByteFromROMMemory(char * ptr)
{
    return *(ptr);
} // GLCD_ReadByteFromROMMemory
```

GLCD_buffer.h

```
// Definitions
//------------------------------
#define DIG_SHIFT               48          // First number in ASCII table
//------------------------------

// Structures
//------------------------------
                                // Screen Variable structure
typedef struct packed_message {
      unsigned  int     mess1;              // 16 bit data
      unsigned  int     mess2;              // 16 bit data
      unsigned  int     mess3;              // 16 bit data
      unsigned  int     mess4;              // 16 bit data
     unsigned  char    locp;               // Location on screen X (0..20/0..31)
            unsigned  char      locs;       // Location on screen Y (0..7/0..7)
            unsigned  char      type;       // 1 - text, 2 - uint, 3 - sint, 4 - psevdo Vbar bar,
                                            // 5 - pseudo Vlimb limb, 6 - pseudo Flag
            unsigned  char  pnt;            // Point location
            unsigned  char  ctrl;           // 0 - empty, 1 - normal, 2 - negative,
                                            // 3 - negative blinking, 4 - normal blinking
     unsigned  char    size;               // Length (number of positions)
} packed_mess;

typedef union {                             // Screen variable union
            unsigned flag       :1;         // If flag type
            packed_mess     attr;           // General field
            char                str[8];     // If string
            unsigned int uint_num;          // If unsigned int
            signed   int   sint_num;        // If signed int
            float           flt_num;        // if float
} packed_mess_types;
//------------------------------

// Functions
//------------------------------

void GLCD_Screen1_Var_Ini (void);                   // Initialize global screen variables

void GLCD_NextByte_Seq (void);                      // Sequentional output next byte to GLCD kontroller

unsigned char ReadROMByte(char * ptr);              // Reads byte from ROM location
                                                    // * ptr - pointer to ROM location element

void GLCD_WriteCharBuff(unsigned char str_,         // Writes text char to RAM video buffer
                        unsigned char pos_,         // str_, pos_ - location, charToWrite - symbol ASCII
                        char charToWrite);

void GLCD_WriteDigBuff(unsigned char str_,          // Writes digit (4x8) char to RAM video buffer
            unsigned char pos_,                     // str_, pos_ - location, charToWrite - symbol ASCII/digit list + signs
                        char charToWrite);

void GLCD_WritePsedBuff     (unsigned char str_,    // Writes pseudo symbol to RAM video buffer
                        unsigned char pos_,         // str_, pos_ - location, charToWrite - symbol from ROM table
                         char charToWrite,          // size_ - size of symbol in bytes
                         unsigned char size_);

void GLCD_WriteStringBuff(unsigned char str_,       // Writes string of ASCII symbols to RAM video buffer
                         unsigned char pos_,        // str_, pos_ - location, * stringToWrite - pointer to 1st element in string
                          char * stringToWrite);

void GLCD_WriteVariables(void);                     // Sequentional graphic generator of Screen Variables
//------------------------------
```

GLCD_buffer.c

```
// Include headers
//-----------------------------------
#include "p30f6010A.h"
#include "general.h"
#include "string.h"
#include "GLCD_buffer.h"
#include "GLCD.h"
#include "font5x8.h"
#include "Scr1.h"
#include "math.h"
//-----------------------------------
```

```
// External variables and functions
//-----------------------------------
extern        unsigned char         ISR_state_machine;                    // Task switch
extern  unsigned char               Timer_1s_flag;                        // 1s flag for blinking etc
extern  void GLCD_GoTo(unsigned char x, unsigned char y);  // Go to to GLCD position
//-----------------------------------

// Internal variables
//-----------------------------------
packed_mess_types   Scr1Var[VAR_NUMBER];     // Screen variables

                                            // Screen buffers:
volatile unsigned char mem_buffer [1024];                   // RAM video buffer
unsigned char * mem_loc         = mem_buffer;               // Pointer to first element of RAM video buffer
unsigned char * Scr1_loc        = Scr1;                     // Pointer to first element of Static Screen 1 ROM

                                            // GLCD routine variables:
unsigned int            mem_count;                          // Used in output to RAM to track current position

unsigned char           GLCD_state_machine;                 // Sequential counter for output to GLCD function
unsigned char           GLCDv_state_machine;                // Sequential counter for graphic gnerator function

unsigned char           cur_pos;                            // Current symbol position
unsigned int            cur_uint;                           // Lock integer
signed   int            cur_sint;                           // Lock signed integer
static unsigned int cur_uint_scaled;                        // Current scaled value (for bar and limb)
unsigned char           displ_ctrl;                         // Display control flag

unsigned char           str, pos;                           // String and position (0..7/0..31)
static unsigned char pnt_shift;                             // Point position (< size attr)
//-----------------------------------

// Functions
//-----------------------------

// Initialize global screen variables
void GLCD_Screen1_Var_Ini (void)
{
   Scr1Var[0].attr.type = 3;                        // dn_Vref - Reference speed/num
   Scr1Var[0].sint_num  = 0;                        //  Speed (-5000...+5000)
   Scr1Var[0].attr.locp = 0;
   Scr1Var[0].attr.locs = 6;
   Scr1Var[0].attr.size = 5;
   Scr1Var[0].attr.pnt =  0;
   Scr1Var[0].attr.ctrl = 1;

   Scr1Var[1].attr.type = 5;                        // db_Vref - Reference speed/bar
   Scr1Var[1].uint_num  = 64;                       // 0 Speed (0 = -100, 127 = +100)
   Scr1Var[1].attr.locp = 0;
   Scr1Var[1].attr.locs = 5;
   Scr1Var[1].attr.size = 4;
   Scr1Var[1].attr.pnt =  0;
   Scr1Var[1].attr.ctrl = 1;

   Scr1Var[2].attr.type = 2;                        // dn_P - Proportional coefficient
   Scr1Var[2].uint_num  = 15;                       // 1000 coeff (00.0 .. 99.9)
   Scr1Var[2].attr.locp = 10;
   Scr1Var[2].attr.locs = 1;
   Scr1Var[2].attr.size = 4;
   Scr1Var[2].attr.pnt =  2;
   Scr1Var[2].attr.ctrl = 1;

   Scr1Var[3].attr.type = 2;                        // dn_I - Integral coefficient
   Scr1Var[3].uint_num  = 22;                       // 10 Speed (0.00 .. 9.99)
   Scr1Var[3].attr.locp = 10;
   Scr1Var[3].attr.locs = 2;
   Scr1Var[3].attr.size = 4;
   Scr1Var[3].attr.pnt =  3;
   Scr1Var[3].attr.ctrl = 1;

   Scr1Var[4].attr.type = 2;                        // dn_D - Differential coefficient/num
   Scr1Var[4].uint_num  = 0;                        // 10 Speed (0..9999)
   Scr1Var[4].attr.locp = 10;
   Scr1Var[4].attr.locs = 3;
   Scr1Var[4].attr.size = 4;
   Scr1Var[4].attr.pnt =  3;
   Scr1Var[4].attr.ctrl = 1;
```

```
Scr1Var[5].attr.type = 6;                    // df_Fb - feedback/flag
Scr1Var[5].flag  =      1;                   // 0 - open loop, 1 -closed loop
Scr1Var[5].attr.locp = 6;
Scr1Var[5].attr.locs = 4;
Scr1Var[5].attr.size = 1;
Scr1Var[5].attr.pnt =  0;
Scr1Var[5].attr.ctrl = 1;

Scr1Var[6].attr.type = 1;                    // dt_SS - System status/text
SS_ld                                        // Message
Scr1Var[6].attr.locp = 4;
Scr1Var[6].attr.locs = 6;
Scr1Var[6].attr.size = 7;
Scr1Var[6].attr.pnt =  0;
Scr1Var[6].attr.ctrl = 1;

Scr1Var[7].attr.type = 2;                    // dn_Ld - load/num
Scr1Var[7].uint_num  = 0;                    // 0 N/m
Scr1Var[7].attr.locp = 19;
Scr1Var[7].attr.locs = 2;
Scr1Var[7].attr.size = 4;
Scr1Var[7].attr.pnt =  0;
Scr1Var[7].attr.ctrl = 0;

Scr1Var[8].attr.type = 4;                    // db_Ld - load/bar
Scr1Var[8].uint_num  = 50;                   // 0 N/m
Scr1Var[8].attr.locp = 24;
Scr1Var[8].attr.locs = 2;
Scr1Var[8].attr.size = 2;
Scr1Var[8].attr.pnt =  0;
Scr1Var[8].attr.ctrl = 0;

Scr1Var[9].attr.type = 3;                    // dn_Vm - measured speed/num
Scr1Var[9].sint_num =  0;                    // 0 r/m
Scr1Var[9].attr.locp = 18;
Scr1Var[9].attr.locs = 4;
Scr1Var[9].attr.size = 5;
Scr1Var[9].attr.pnt =  0;
Scr1Var[9].attr.ctrl = 1;

Scr1Var[10].attr.type = 4;                   // db_Vm - measured speed/bar
Scr1Var[10].uint_num  = 50;                  // 0 r/m
Scr1Var[10].attr.locp = 24;
Scr1Var[10].attr.locs = 4;
Scr1Var[10].attr.size = 2;
Scr1Var[10].attr.pnt =  0;
Scr1Var[10].attr.ctrl = 1;

Scr1Var[11].attr.type = 6;                   // df_P - Pause/flag
Scr1Var[11].flag  =  0;                      // 0 - idle, 1 -pause
Scr1Var[11].attr.locp = 20;
Scr1Var[11].attr.locs = 6;
Scr1Var[11].attr.size = 1;
Scr1Var[11].attr.pnt =  0;
Scr1Var[11].attr.ctrl = 1;

Scr1Var[12].attr.type = 6;                   // df_S - Start
Scr1Var[12].flag  =  0;                      // 0 - idle, 1 -start
Scr1Var[12].attr.locp = 23;
Scr1Var[12].attr.locs = 6;
Scr1Var[12].attr.size = 1;
Scr1Var[12].attr.pnt =  0;
Scr1Var[12].attr.ctrl = 4;

Scr1Var[13].attr.type = 2;                   // dn_Um - measured voltage/num
Scr1Var[13].uint_num  = 0;                   // 0 V
Scr1Var[13].attr.locp = 26;
Scr1Var[13].attr.locs = 2;
Scr1Var[13].attr.size = 4;
Scr1Var[13].attr.pnt =  2;                   // 00.0
Scr1Var[13].attr.ctrl = 1;

Scr1Var[14].attr.type = 4;                   // db_Um - measured voltage/bar
Scr1Var[14].uint_num  = 0;                   // 0..1023 (0 V)
Scr1Var[14].attr.locp = 31;
Scr1Var[14].attr.locs = 2;
Scr1Var[14].attr.size = 2;
Scr1Var[14].attr.pnt =  0;
Scr1Var[14].attr.ctrl = 1;
```

```
  Scr1Var[15].attr.type = 2;                    // dn_Im - measured current/num
  Scr1Var[15].uint_num  = 0;                    // 0 A
  Scr1Var[15].attr.locp = 26;
  Scr1Var[15].attr.locs = 4;
  Scr1Var[15].attr.size = 4;
  Scr1Var[15].attr.pnt =  3;                    // 0.00
  Scr1Var[15].attr.ctrl = 1;

  Scr1Var[16].attr.type = 4;                    // db_Im - measured current/bar
  Scr1Var[16].uint_num  = 0;                    // 0 A
  Scr1Var[16].attr.locp = 31;
  Scr1Var[16].attr.locs = 4;
  Scr1Var[16].attr.size = 2;
  Scr1Var[16].attr.pnt =  0;
  Scr1Var[16].attr.ctrl = 0;

  Scr1Var[17].attr.type = 2;                    // dn_Tm - measured temperature/num
  Scr1Var[17].uint_num  = 0;                    // 0 C
  Scr1Var[17].attr.locp = 26;
  Scr1Var[17].attr.locs = 6;
  Scr1Var[17].attr.size = 4;
  Scr1Var[17].attr.pnt =  2;                    // 00.0
  Scr1Var[17].attr.ctrl = 1;

  Scr1Var[18].attr.type = 4;                    // db_Im - measured temperature/bar
  Scr1Var[18].uint_num  = 0;                    // 0 C
  Scr1Var[18].attr.locp = 31;
  Scr1Var[18].attr.locs = 6;
  Scr1Var[18].attr.size = 2;
  Scr1Var[18].attr.pnt =  0;
  Scr1Var[18].attr.ctrl = 1;

  GLCDv_state_machine = 0;
} // GLCD_Screen1_Var_Ini


// Output next byte to GLCD controller
void GLCD_NextByte (void)
{
  GLCD_GoTo(mem_count%128,mem_count/128);
  GLCD_WriteData(*(mem_loc+mem_count));
  mem_count++;
  mem_count %= 1024;
} // GLCD_NextByte


// Sequentional output next byte to GLCD controller
void GLCD_NextByte_Seq (void)
{
  switch (GLCD_state_machine)
  {
        case 1:
              GLCD_GoTo(mem_count%128,mem_count/128);
        break;

        case 2:
              GLCD_WriteData(*(mem_loc+mem_count) | *(Scr1_loc+mem_count));
        break;

        case 3:
              mem_count++;
              mem_count %= 1024;
        break;

        case 4:
              GLCD_state_machine = 0;
              if (mem_count%32 == 0) ISR_state_machine++;  // Number of bytes to output per ISR
        break;

        default:
        break;
  } // SWITCH
GLCD_state_machine++;
} // GLCD_NextByte_Seq
```

```
// Reads byte from ROM location
// * ptr - pointer to ROM location element
unsigned char ReadROMByte(char * ptr)
{
   return *(ptr);
} // ReadROMByte


// Writes text char to RAM video buffer
// str_, pos_ - location, charToWrite - symbol ASCII
void GLCD_WriteCharBuff(unsigned char str_, unsigned char pos_, char charToWrite)
{
   int i;
   str = str_; pos= pos_;
   charToWrite -= 32;
   for(i = 0; i < 6; i++)
   {

            switch (displ_ctrl)
            {
                  case 0:
                        mem_buffer[str*128+pos*6+i] = 0;
                  break;

                  case 1:
                        mem_buffer[str*128+pos*6+i] = (ReadROMByte((char *)((int)font5x8 + (6 * charToWrite) + i)));
                  break;

                  case 2:
                        mem_buffer[str*128+pos*6+i] = ~(ReadROMByte((char *)((int)font5x8 + (6 * charToWrite) + i)));

                  default:
                  break;
            } // close switch
   } // close for
} // GLCD_WriteCharBuff


// Writes digit (4x8) char to RAM video buffer
// str_, pos_ - location, charToWrite - symbol ASCII/digit list + signs
void GLCD_WriteDigBuff(unsigned char str_, unsigned char pos_, char charToWrite)
{
   int i;
   str = str_; pos= pos_;
   charToWrite -= 43;

   for(i = 0; i < 4; i++)
   {

            switch (displ_ctrl)
            {
                  case 0:
                        mem_buffer[str*128+pos*4+i] = Scr1[str*128+pos*4+i];
                  break;

                  case 1:
                        mem_buffer[str*128+pos*4+i] =
                        ((ReadROMByte((char *)((int)font4x8 + (4 * charToWrite) + i))));
                  break;

                  case 2:
                        mem_buffer[str*128+pos*4+i] = (~(ReadROMByte((char *)((int)font4x8 +
            (4 * charToWrite) + i)))) & 0b11111110;

                  default:
                  break;
            } // close switch
   } // close for
} // GLCD_WriteDigBuff
```

```
// Writes pseudo symbol to RAM video buffer
// str_, pos_ - location, charToWrite - symbol from ROM table
// size_ - size of symbol in bytes
void GLCD_WritePsedBuff       (unsigned char str_, unsigned char pos_, char charToWrite, unsigned char size_)
{
   int i;
   str = str_; pos= pos_;
   for(i = 0; i < size_; i++)
   {
            switch (displ_ctrl)
            {
                     case 0:
                              mem_buffer[str*128+pos*size_+i] = 0;
                     break;

                     case 1:
                              mem_buffer[str*128+pos*size_+i] = (ReadROMByte((char *)((int)psed4x8 + (size_ * charToWrite) + i)));
                     break;

                     case 2:
                              mem_buffer[str*128+pos*size_+i] = ~(ReadROMByte((char *)((int)psed4x8 + (size_ * charToWrite) + i)));

                     default:
                     break;
            } // close switch
   } // close for
} // GLCD_WritePsedBuff


// Writes string of ASCII symbols to RAM video buffer
// str_, pos_ - location, * stringToWrite - pointer to 1st element in string
void GLCD_WriteStringBuff(unsigned char str_, unsigned char pos_, char * stringToWrite)
{
   str = str_; pos= pos_;
   while(*stringToWrite)
   {
            GLCD_WriteCharBuff(str, pos, *stringToWrite++);
            pos++;
   } // close while
} // GLCD_WriteStringBuff


// Sequentional graphic generator of Screen Variables
void GLCD_WriteVariables(void)
{
   switch (Scr1Var[GLCDv_state_machine].attr.type)                                    // "Attr" SWITCH
    {
            case 1:           // Normal text 5x7
                     if (cur_pos<Scr1Var[GLCDv_state_machine].attr.size)     // "Size" IF
                     {
                              switch (Scr1Var[GLCDv_state_machine].attr.ctrl)          // "Ctrl" SWITCH
                              {
                                       case 0:
                                                displ_ctrl = 0;                    // No output
                                       break;

                                       case 1:
                                                displ_ctrl = 1;                    // Normal output
                                       break;

                                       case 2:
                                                displ_ctrl = 2;                    // Negative output
                                       break;

                                   case 3:                                         // Blinking negative
                                                displ_ctrl = (Timer_1s_flag & 0b00000001)+1;
                                       break;

                                       case 4:                                         // Blinking normal
                                                displ_ctrl = (Timer_1s_flag & 0b00000001);

                                       default:
                                       break;
                              } // close "Ctrl" SWITCH
                                                                                       // Point generator
                              if ((cur_pos+1) == Scr1Var[GLCDv_state_machine].attr.pnt) // "Pnt" IF
                              {
                                       GLCD_WriteCharBuff (Scr1Var[GLCDv_state_machine].attr.locs,
                                                                (Scr1Var[GLCDv_state_machine].attr.locp+cur_pos),
```

```
                                                                                '.');
                        pnt_shift = 1;
                } // close "Pnt" IF

                GLCD_WriteCharBuff (Scr1Var[GLCDv_state_machine].attr.locs,
                                                (Scr1Var[GLCDv_state_machine].attr.locp+cur_pos+pnt_shift),
                                                        Scr1Var[GLCDv_state_machine].str[cur_pos]);
                cur_pos++;                                      // Next element...
                GLCDv_state_machine--;                          // Advance GLCDv machine
break;
        } // close "Size" IF

        displ_ctrl = 0;
        pnt_shift = 0;
        cur_pos = 0;
        break;

case 2:         // Unsigned int numbers...
        if ((cur_pos == Scr1Var[GLCDv_state_machine].attr.pnt) &&       // "Pnt" IF
                (Scr1Var[GLCDv_state_machine].attr.pnt <= Scr1Var[GLCDv_state_machine].attr.size) &&
                (Scr1Var[GLCDv_state_machine].attr.pnt != 0))
        {
                GLCD_WriteDigBuff (Scr1Var[GLCDv_state_machine].attr.locs,
                                                (Scr1Var[GLCDv_state_machine].attr.locp+
                                                (Scr1Var[GLCDv_state_machine].attr.size - cur_pos)),
                                                                                        '.');
                pnt_shift = 1;

        } // close "Pnt" IF

        if (cur_pos<=Scr1Var[GLCDv_state_machine].attr.size-pnt_shift)      // "Pnt" IF
        {
                switch (cur_pos)
        // "Cur_pos" SWITCH
                {
                        case 0:
                                cur_uint = Scr1Var[GLCDv_state_machine].uint_num;
                                switch (Scr1Var[GLCDv_state_machine].attr.ctrl)         // "Ctrl" SWITCH
                                {
                                        case 0:
                                                displ_ctrl = 0;
                                        break;

                                        case 1:
                                                displ_ctrl = 1;
                                        break;

                                        case 2:
                                                displ_ctrl = 2;
                                        break;

                                        case 3:
                                                displ_ctrl = (Timer_1s_flag & 0b00000001)+1;
                                        break;

                                        case 4:
                                                 displ_ctrl = (Timer_1s_flag & 0b00000001);

                                        default:
                                        break;
                                } // close "Ctrl" SWITCH
                        break;

                        case 1:
                                GLCD_WriteDigBuff (Scr1Var[GLCDv_state_machine].attr.locs,
                                                (Scr1Var[GLCDv_state_machine].attr.locp+
                                                (Scr1Var[GLCDv_state_machine].attr.size - cur_pos-pnt_shift)),
                                                 (cur_uint)%10+DIG_SHIFT);
                        break;

                        case 2:
                                GLCD_WriteDigBuff (Scr1Var[GLCDv_state_machine].attr.locs,
                                                  Scr1Var[GLCDv_state_machine].attr.locp+
                                                (Scr1Var[GLCDv_state_machine].attr.size - cur_pos-pnt_shift),
                                                ((cur_uint)/10)%10+DIG_SHIFT);
                        break;
```

```
                                case 3:
                                        GLCD_WriteDigBuff (Scr1Var[GLCDv_state_machine].attr.locs,
                                                           Scr1Var[GLCDv_state_machine].attr.locp+
                                                          (Scr1Var[GLCDv_state_machine].attr.size - cur_pos-pnt_shift),
                                                          ((cur_uint)/100)%10+DIG_SHIFT);
                                        break;

                                        case 4:
                                        GLCD_WriteDigBuff (Scr1Var[GLCDv_state_machine].attr.locs,
                                                           Scr1Var[GLCDv_state_machine].attr.locp+
                                                           (Scr1Var[GLCDv_state_machine].attr.size - cur_pos-pnt_shift),
                                                          ((cur_uint)/1000)%10+DIG_SHIFT);
                                        break;

                                        case 5:
                                        GLCD_WriteDigBuff (Scr1Var[GLCDv_state_machine].attr.locs,
                                                           Scr1Var[GLCDv_state_machine].attr.locp+
                                                           (Scr1Var[GLCDv_state_machine].attr.size - cur_pos-pnt_shift),
                                                           (cur_uint)/10000+DIG_SHIFT);

                                        default:
                                        break;
                        } //  close "Cur_pos" SWITCH

                        GLCDv_state_machine--;
                        cur_pos++;
break;
        } // close "Pnt" IF

        displ_ctrl = 0;
        pnt_shift = 0;
        cur_pos=0;
break;

case 3:         // Signed int numbers...
        if ((cur_pos == Scr1Var[GLCDv_state_machine].attr.pnt) &&
                (Scr1Var[GLCDv_state_machine].attr.pnt < Scr1Var[GLCDv_state_machine].attr.size) &&
                (Scr1Var[GLCDv_state_machine].attr.pnt != 0))
        {
                GLCD_WriteDigBuff (Scr1Var[GLCDv_state_machine].attr.locs,
                                                           (Scr1Var[GLCDv_state_machine].attr.locp+
                                                           (Scr1Var[GLCDv_state_machine].attr.size - cur_pos)),
                                                                                                   '.');
                pnt_shift = 1;
        }

        if (cur_pos <= Scr1Var[GLCDv_state_machine].attr.size-pnt_shift-1)  // "Cur_pos" IF
        {
                switch (cur_pos)
        // "Cur_pos" SWITCH
                {
                        case 0:
                                cur_sint = Scr1Var[GLCDv_state_machine].sint_num;
                                switch (Scr1Var[GLCDv_state_machine].attr.ctrl)
                                {
                                        case 0:
                                                displ_ctrl = 0;
                                        break;

                                        case 1:
                                                displ_ctrl = 1;
                                        break;

                                        case 2:
                                                displ_ctrl = 2;
                                        break;

                                        case 3:
                                                displ_ctrl = (Timer_1s_flag & 0b00000001)+1;
                                        break;

                                        case 4:
                                                displ_ctrl = (Timer_1s_flag & 0b00000001);

                                        default:
                                        break;
                                }
```

```
                                        if (cur_sint > 0)
                                        {
                                                GLCD_WriteDigBuff (Scr1Var[GLCDv_state_machine].attr.locs,
(Scr1Var[GLCDv_state_machine].attr.locp+cur_pos-pnt_shift),
                                                                                                        '+');
                                        }
                                        else if (cur_sint < 0)
                                        {
                                                GLCD_WriteDigBuff (Scr1Var[GLCDv_state_machine].attr.locs,
(Scr1Var[GLCDv_state_machine].attr.locp+cur_pos-pnt_shift),
                                                                                                        '-');
                                        }

                                        else
                                        {
                                                GLCD_WriteDigBuff (Scr1Var[GLCDv_state_machine].attr.locs,
                        (Scr1Var[GLCDv_state_machine].attr.locp+cur_pos-pnt_shift),
                                                                                                         '0');
                                        }

                                        cur_sint = fabs(cur_sint);
                                        break;

                                        case 5:
                                                GLCD_WriteDigBuff (Scr1Var[GLCDv_state_machine].attr.locs,
                                                 (Scr1Var[GLCDv_state_machine].attr.locp+
                                                Scr1Var[GLCDv_state_machine].attr.size - cur_pos-pnt_shift),
                                                (cur_sint/10000)+DIG_SHIFT);
                                        break;

                                        case 4:
                                                GLCD_WriteDigBuff (Scr1Var[GLCDv_state_machine].attr.locs,
                                                (Scr1Var[GLCDv_state_machine].attr.locp+
Scr1Var[GLCDv_state_machine].attr.size - cur_pos-pnt_shift),
                                                ((cur_sint)/1000)%10+DIG_SHIFT);
                                        break;

                                        case 3:
                                                GLCD_WriteDigBuff (Scr1Var[GLCDv_state_machine].attr.locs,
                                                 (Scr1Var[GLCDv_state_machine].attr.locp+
                                                 Scr1Var[GLCDv_state_machine].attr.size - cur_pos-pnt_shift),
                                                ((cur_sint)/100)%10+DIG_SHIFT);
                                        break;

                                        case 2:
                                                GLCD_WriteDigBuff (Scr1Var[GLCDv_state_machine].attr.locs,
                                                (Scr1Var[GLCDv_state_machine].attr.locp+
                                                Scr1Var[GLCDv_state_machine].attr.size - cur_pos-pnt_shift),
                                                ((cur_sint)/10)%10+DIG_SHIFT);
                                        break;

                                        case 1:
                                                GLCD_WriteDigBuff (Scr1Var[GLCDv_state_machine].attr.locs,
                                                (Scr1Var[GLCDv_state_machine].attr.locp+
                                                 Scr1Var[GLCDv_state_machine].attr.size - cur_pos-pnt_shift),
                                                (cur_sint)%10+DIG_SHIFT);
                                        break;

                                        default:
                                        break;
                        } // close "Cur_pos" SWITCH
                        GLCDv_state_machine--;
                        cur_pos++;
        break;
                } // close "Cur_pos" IF
                displ_ctrl = 0;
                pnt_shift = 0;
                cur_pos=0;
        break;

        case 4:                 // Bar graph - graphics (0...1023)
                if (cur_pos <= Scr1Var[GLCDv_state_machine].attr.size)      // "Size" IF
                {
                        switch (cur_pos)
                                                                                // "Cur_pos" SWITCH
                        {
                                case 0:
                                        cur_uint = Scr1Var[GLCDv_state_machine].uint_num;
                                        cur_uint_scaled = (cur_uint*Scr1Var[GLCDv_state_machine].attr.size)>>8;
```

```
                                        switch (Scr1Var[GLCDv_state_machine].attr.ctrl)
                                        {
                                                case 0:
                                                    displ_ctrl = 0;
                                                break;

                                                case 1:
                                                        displ_ctrl = 1;
                                                break;

                                                case 2:
                                                        displ_ctrl = 2;
                                                break;

                                                case 3:
                                                        displ_ctrl = (Timer_1s_flag & 0b00000001)+1;
                                                break;

                                                case 4:
                                                        displ_ctrl = (Timer_1s_flag & 0b00000001);

                                                default:
                                                break;
                                        } // close SWITCH
                                break;

                                default:
                                        if (cur_uint_scaled < 4)
                                        {
                                        GLCD_WritePsedBuff (Scr1Var[GLCDv_state_machine].attr.locs - cur_pos+1,
                                                (Scr1Var[GLCDv_state_machine].attr.locp),
                                                 (cur_uint_scaled+7),4);
                                                cur_uint_scaled = 0;
                                        }

                                        else
                                        {
                                                cur_uint_scaled -= 4;
                                        GLCD_WritePsedBuff (Scr1Var[GLCDv_state_machine].attr.locs –
                                        cur_pos+1,
                                                (Scr1Var[GLCDv_state_machine].attr.locp), (11),4);
                                        }

                                        break;

                        } // close "Cur_pos" SWITCH

                        GLCDv_state_machine--;
                        cur_pos++;
break;
                }  // close "Size" IF

                displ_ctrl = 0;
                cur_pos=0;
break;

case 5:                 // Regullator/limb - graphic (0..1023)
                if (cur_pos <= Scr1Var[GLCDv_state_machine].attr.size)
                {
                        switch (cur_pos)  // "Cur_pos" SWITCH
                        {
                                case 0:
                                        cur_uint = Scr1Var[GLCDv_state_machine].uint_num;
                                        cur_uint_scaled = (cur_uint*Scr1Var[GLCDv_state_machine].attr.size)>>8;
                                        switch (Scr1Var[GLCDv_state_machine].attr.ctrl)
                                        {
                                                case 0:
                                                        displ_ctrl = 0;
                                                break;

                                                case 1:
                                                        displ_ctrl = 1;
                                                break;

                                                case 2:
                                                        displ_ctrl = 2;
                                                break;

                                                case 3:
                                                        displ_ctrl = (Timer_1s_flag & 0b00000001)+1;
```

```
                                                break;

                                                case 4:
                                                        displ_ctrl = (Timer_1s_flag & 0b00000001);

                                                default:
                                                break;
                                        }
                                break;

                                default:
                                        if (cur_uint_scaled < 4)
                                        {
                                                GLCD_WritePsedBuff (Scr1Var[GLCDv_state_machine].attr.locs –
                                                cur_pos+1,
                                                 (Scr1Var[GLCDv_state_machine].attr.locp),
                                                 (cur_uint_scaled+7+6),4);
                                                cur_uint_scaled = 32;
                                        }

                                        else
                                        {
                                                cur_uint_scaled -= 4;
                                                GLCD_WritePsedBuff (Scr1Var[GLCDv_state_machine].attr.locs –
                                                cur_pos+1,
                                                (Scr1Var[GLCDv_state_machine].attr.locp), (11+1),4);
                                        }
                                break;

                        } // close "Cur_pos" SWITCH
                        GLCDv_state_machine--;
                        cur_pos++;
        break;
                        }
                        displ_ctrl = 0;
                        cur_pos=0;
        break;

        case 6:         // Flag - bit (0/1)
                switch (Scr1Var[GLCDv_state_machine].attr.ctrl)
                {
                        case 0:
                                        displ_ctrl = 0;
                        break;

                        case 1:
                                        displ_ctrl = 1;
                        break;

                        case 2:
                                        displ_ctrl = 2;
                        break;

                        case 3:
                                        displ_ctrl = (Timer_1s_flag & 0b00000001)+1;
                        break;

                        case 4:
                                        displ_ctrl = (Timer_1s_flag & 0b00000001);

                        default:
                        break;
                }

                GLCD_WritePsedBuff (Scr1Var[GLCDv_state_machine].attr.locs,
                                                (Scr1Var[GLCDv_state_machine].attr.locp),
                                                (17+Scr1Var[GLCDv_state_machine].flag),4);

                displ_ctrl = 0;

        default:
        break;
  }

GLCDv_state_machine++;
if (GLCDv_state_machine>=VAR_NUMBER) GLCDv_state_machine = 0;
//ISR_state_machine++;
}
```

Interrupts.h

```
// Functions
//-----------------------------
void Init_Timer1(void);                                 // Timer1 initialization

void __attribute__((__interrupt__)) _T1Interrupt(void);  // Timer1 ISR
//-----------------------------
```

Interrupts.c

```
// Include headers
//-----------------------------
#include "p30f6010A.h"
#include "DirectLEDs.h"
#include "GLCD_buffer.h"
#include "general.h"
//-----------------------------

// Variables, accesible from outside
//-----------------------------------
unsigned char           ISR_state_machine;      // BG task state machine
unsigned char           Timer_1s_flag;          // Used for text blinking;
unsigned long           TempDisplayAvrSpeed;    // Averaged (1.6s) speed
//-----------------------------------

// External variables and functions
//-----------------------------------
extern void GLCD_WriteVariables(void);                  // Convert variables to graphic and put to video RAM
extern void UpdateReadMax6959_Seq(void);                // Update Max6959 7seg+LEDs+read buttons
extern void GLCD_NextByte_Seq(void);                    // Output next 1 byte to GLCD screen
extern unsigned long adc_buffer[6];                     // ADC buffer
extern packed_mess_types Scr1Var[VAR_NUMBER];           // Screen 1 variables array
extern unsigned int MonitorBufferCount;                 // Monitor buffer count var
extern unsigned int CurrentBufferCount;                 // Current adc buffer count var
extern flags_packed  Flags;                             // Control flags
//-----------------------------------

// Internal variables and functions
//-----------------------------------
static unsigned int Timer_1s_count;                     // Counter for text blinking
//-----------------------------------

// Functions
//-----------------------------------

// Timer1 initialization
void Init_Timer1(void)
{
  ISR_state_machine = 0;        // Reset ISR state machine

  T1CON = 0;                    // Timer1 to reset state
  IPC0bits.T1IP = 4;            // Set Timer1 I priority level to 4

  PR1 = 700;                    // Set Timer1 period register (LCD/LEDs/buttons update)
  T1CON = 0x8000;               // Timer1 ON, prescaler 1:1 Tcy

  IFS0bits.T1IF = 0;            // Reset Timer1 I flag
  IEC0bits.T1IE = 1;            // Enable Timer1 I
} // Init_Timer1


// Timer1 ISR
void __attribute__((__interrupt__)) _T1Interrupt( void )
{
 if (ISR_state_machine < 1)
{
  UpdateReadMax6959_Seq();              // Read/Update MAX
}

else if (ISR_state_machine < 2)
  {
        GLCD_NextByte_Seq();        // Write 32 bytes to GLCD +
                                    // + Write to RAM VAR_NUM variables
  }

else
  {
        Timer_1s_count++;           // Blinking + extra fun. generator
        if (!(Timer_1s_count%64))   // Display speed
```

```
                {
                        if (Flags.RotDirection == 1)
                        {
                                dn_Vm = TempDisplayAvrSpeed;
                                db_Vm = (TempDisplayAvrSpeed/5)+35;
                        }
                        else if (Flags.RotDirection == 2)
                        {
                                dn_Vm = (-TempDisplayAvrSpeed);
                                db_Vm = (TempDisplayAvrSpeed/5)+35;
                        }
                        else {dn_Vm = 0; db_Vm = 0;}

                        Timer_1s_flag = ~Timer_1s_flag;

                } // close if

        if  (!(Timer_1s_count%128))                                     // Estimate, Display current
        {
                if ((Flags.RotDirection) && (CurrentBufferCount == 1024))
                {
                        dn_Im = ((unsigned long)((adc_buffer[1]/2096)-243));
                        CurrentBufferCount = 0;
                        adc_buffer[1] = 0;
                }
                else if (!(Flags.RotDirection))
                {
                        dn_Im = 0;
                        CurrentBufferCount = 0;
                        adc_buffer[1] = 0;
                }
        }

        if ((MonitorBufferCount == 256) && !(Timer_1s_count%256))
        {
                                                                        // Estimate, Display Temperature, Bus voltage
                dn_Um = (adc_buffer[5]/282);
                db_Um = dn_Um;

                adc_buffer[0] = adc_buffer[0]>>8;
                adc_buffer[0] = (unsigned long)((1233450 - adc_buffer[0]*(2188 - adc_buffer[0]))/1000);
                dn_Tm = (unsigned int)adc_buffer[0];
                db_Tm = dn_Tm;

                adc_buffer[0] = 0;
                adc_buffer[5] = 0;
                MonitorBufferCount = 0;
        }

        ISR_state_machine = 0;
  }

TMR1                = 0;        // Release main for N more cycles
IFS0bits.T1IF       = 0;        // Reset Timer 1 interrupt flag
} // _T1Interrupt
```

KS0108.h

```
// Definitions
//-----------------------------
                                // GLCD definitions
#define DISPLAY_SET_Y           0x40    // Command to set GLCD Y locationa
#define DISPLAY_SET_X           0xB8    // Command to set GLCD X locationa
#define DISPLAY_START_LINE      0xC0    // GLCD set start line
#define DISPLAY_ON_CMD          0x3E    // GLCD switch ON display
#define DISPLAY_STATUS_BUSY     0x80    // Display status get
//-----------------------------

// Functions
//-----------------------------
void GLCD_Initialize(void);                             // Initialize GLCD screen

void GLCD_GoTo(unsigned char, unsigned char);           // Set current memory location for GLCD
                                                        // x,y - location (128x64)

void GLCD_ClearScreen(void);                            // Clear GLCD screen

void GLCD_SetPixel(unsigned char x,                     // Set GLCD pixel to 0 or 1
                   unsigned char y,                     // x,y - location; color - 1/0 pixel
                   unsigned char color);

void GLCD_Bitmap(char *, unsigned char,                 // Send part of bitmap image from ROM to GLCD
                 unsigned char,                         // * bmp - pointer to image begin,
                 unsigned char,                         // x,y - coordinates of up-left point
                 unsigned char);                        // dx,dy - size of block

//-----------------------------
```

KS0108.c

```
// System include headers
//-----------------------------
#include "general.h"
#include "GLCD.h"
#include "KS0108.h"
//-----------------------------

// Variables, accesible from outside
//-----------------------------------
unsigned char screen_x = 0, screen_y = 0;               // Pixel coordinates
//-----------------------------------

// External variables and functions
//-----------------------------------
extern void GLCD_InitializePorts(void);                 // Initialize ports for KS0108 op
//-----------------------------------

// Functions
//-----------------------------------

// Initialize GLCD screen
void GLCD_Initialize(void)
{
   unsigned char i;
   GLCD_InitializePorts();                              // Init dsPIC ports
   for(i = 0; i < 2; i++)                               // Switch on CS1 and CS2 parts
              GLCD_WriteCommand((DISPLAY_ON_CMD | ON), i);
   GLCD_GoTo(0,0);                                      // Position to 0:0
} // close GLCD_Initialize


// Set current memory location for GLCD
// x,y - location (128x64)
void GLCD_GoTo(unsigned char x, unsigned char y)
{
   unsigned char i;
   screen_x = x;         screen_y = y;

   for(i = 0; i < KS0108_SCREEN_WIDTH/64; i++)
   {
              GLCD_WriteCommand(DISPLAY_SET_Y | 0,i);
              GLCD_WriteCommand(DISPLAY_SET_X | y,i);
              GLCD_WriteCommand(DISPLAY_START_LINE | 0,i);
   } // close for
```

```
   GLCD_WriteCommand(DISPLAY_SET_Y | (x % 64), (x / 64));
   GLCD_WriteCommand(DISPLAY_SET_X | y, (x / 64));
} // close GLCD_GoTo


// Clear GLCD screen
void GLCD_ClearScreen(void)
{
   unsigned char i, j;
   for(j = 0; j < KS0108_SCREEN_HEIGHT/8; j++)
   {
                GLCD_GoTo(0,j);
                for(i = 0; i < KS0108_SCREEN_WIDTH; i++)
                        GLCD_WriteData(0b00000000);
   }
   GLCD_GoTo(0,0);
} // close GLCD_ClearScreen


// Set GLCD pixel to 0 or 1
// x,y - location; color - 1/0 pixel
void GLCD_SetPixel(unsigned char x, unsigned char y, unsigned char color)
{
   unsigned char tmp;
   GLCD_GoTo(x, (y / 8));                  // Read data at location
   tmp = GLCD_ReadData();
   GLCD_GoTo(x, (y / 8));                  // Set location again (as was advanced)
   tmp = GLCD_ReadData();
   GLCD_GoTo(x, (y / 8));
   tmp |= (1 << (y % 8));                  // OR with what was in location
   GLCD_WriteData(tmp);                    // Update byte
} // close GLCD_SetPixel


// Send part of bitmap image from ROM to GLCD
// * bmp - pointer to image begin, x,y - coordinates of up-left point, dx,dy - size of block
void GLCD_Bitmap(char * bmp, unsigned char x, unsigned char y, unsigned char dx, unsigned char dy)
{
   unsigned char i, j;
   for(j = 0; j < dy / 8; j++)
   {
                GLCD_GoTo(x,y + j);
                for(i = 0; i < dx; i++)
                GLCD_WriteData(GLCD_ReadByteFromROMMemory(bmp++));
   }
} // close GLCD_Bitmap
```

MAX6959.h

```
// Definitions
//------------------------------
                                                    // Macros for I2C Idle state
#define IdleI2C_m      if(I2CCONbits.SEN || I2CCONbits.PEN ||
            I2CCONbits.RCEN || I2CCONbits.ACKEN ||
            I2CSTATbits.TRSTAT) break;
                                                    // Max6959 buttons definitions
#define BUT_SW3             SEGbut.button.S3
#define BUT_SW4             SEGbut.button.S2
#define BUT_SW5             SEGbut.button.S1
#define BUT_SW6             SEGbut.button.S4
#define BUT_SW7             SEGbut.button.S5
#define BUT_SW8             SEGbut.button.S7
#define BUT_SW9             SEGbut.button.S6
#define BUT_SW10            SEGbut.button.S0
                                                    // Max6959 LEDs
#define LED_D3              SEGled.led.D3
#define LED_D4              SEGled.led.D4
#define LED_D11             SEGled.led.D11
#define LED_D12             SEGled.led.D12
                                                    // Decimal point definition
#define LED_DP4             SEGled.led.DP4
#define LED_DP3             SEGled.led.DP3
#define LED_DP2             SEGled.led.DP2
#define LED_DP1             SEGled.led.DP1
//------------------------------

// Structures
//------------------------------
typedef struct packed_bit_buttons {                 // Buttons - bits
      unsigned S0    :1;
      unsigned S1    :1;
      unsigned S2    :1;
      unsigned S3    :1;
      unsigned S4    :1;
      unsigned S5    :1;
      unsigned S6    :1;
      unsigned S7    :1;
} packed_bit_buttons;

typedef union {                                     // Buttons - bits + char union
            unsigned char buttons;                  // Acc: .button.SW1; .buttons
            packed_bit_buttons button;
} packed_buttons;

typedef struct packed_bit_leds {// LEDs - bits
      unsigned D3     :1;
      unsigned D4     :1;
      unsigned D11    :1;
      unsigned D12    :1;
      unsigned DP4    :1;
      unsigned DP3    :1;
      unsigned DP2    :1;
      unsigned DP1    :1;
} packed_bit_leds;

typedef union {                                     // LEDs - bits + char union
            unsigned char leds;                     // Acc: .led.D3; .leds
            packed_bit_leds led;
} packed_leds;
//------------------------------

// Functions
//------------------------------
void InitI2C(void);                                 // Initiate I2C bus

void InitMax6959(void);                             // Initialize MAX6959

void UpdateReadMax6959_Seq (void);    // Sequential Update-Read routine

void UpdateMax6959_Cont (void);                     // Updates LEDs from SEGnum (0.9999)
                                                    // and SEGled (char or bits)
                                                    // Ret: (1) - success, (0) - incorrect num

void ReadMax6959_Cont (void);                       // Reads buttons to SEGbut (char or bits)

void CloseI2C(void);                                // Close I2C session
//------------------------------
```

MAX6959.c

```
// System include headers
//-----------------------------
#include "general.h"
#include "p30f6010A.h"
#include "max6959.h"
#include <i2c.h>
//-----------------------------

// Variables, accesible from outside
//-----------------------------------
unsigned int                SEGnum;          // Number 0000....9999 for Seg Display
volatile packed_leds        SEGled;          // LEDs + decimal point
volatile packed_buttons     SEGbut;          // Buttons
//-----------------------------------

// External variables and functions
//-----------------------------------
extern      unsigned char     ISR_state_machine;     // Task switch
//-----------------------------------

// Internal variables
//-----------------------------------
unsigned char               I2C_state_machine;     // Sequential state machine counter
static unsigned  int    temp;                      // Used to calc digits
//-----------------------------------

// Functions
//-----------------------------------

// Initialize I2C
void InitI2C(void)
{
   unsigned int config2, config1;

    config1 = (I2C_ON & I2C_IDLE_CON & I2C_CLK_HLD     // Configure I2C for 7 bit address mode
          & I2C_IPMI_DIS & I2C_7BIT_ADD
          & I2C_SLW_DIS & I2C_SM_DIS &
          I2C_GCALL_DIS & I2C_STR_DIS &
          I2C_NACK & I2C_ACK_DIS & I2C_RCV_DIS &
          I2C_STOP_DIS & I2C_RESTART_EN
          & I2C_START_DIS);

    config2 = 25;                                      // Baud rate 400 Khz (23-PLL8, 47-PLL16)
    OpenI2C(config1,config2);                          // Open I2C bus
   IdleI2C();
} // close InitI2C


// Initialize MAX6959
void InitMax6959 (void)
{

   StartI2C();                              // Start I2C frame
   while(I2CCONbits.SEN);                   // Wait for Start sequence END

   MasterWriteI2C(0b01110010);              // Send DEVICE CODE + WRITE
   IdleI2C();

   MasterWriteI2C(0x00);                    // Choose No-OP Reg
   IdleI2C();
   MasterWriteI2C(0x00);                    // Send  0x00              (Reg 0x00)
   IdleI2C();

   MasterWriteI2C(0x0F);                    // Decode all digits  (Reg 0x01)
   IdleI2C();
   MasterWriteI2C(0x10);                    // Set intensity 10% (Reg 0x02)
   IdleI2C();
   MasterWriteI2C(0x03);                    // Set scan limit all   (Reg 0x03)
   IdleI2C();
   MasterWriteI2C(0x21);                    // Set normal operation     (Reg 0x04)
   IdleI2C();

    StopI2C();                              // Stop I2C frame
    while(I2CCONbits.PEN);                  // Wait for Stop sequence end

   StartI2C();                              // Start I2C frame
   while(I2CCONbits.SEN);                   // Wait for Start sequence end
```

```
   MasterWriteI2C(0b01110010);                  // Send DEVICE CODE + WRITE
   IdleI2C();

   MasterWriteI2C(0x06);                        // Choose 0x06 Reg
   IdleI2C();
   MasterWriteI2C(0x98);                        // SEG driver + inp1,2 keyscan
   IdleI2C();

    StopI2C();                                  // Stop I2C frame
    while(I2CCONbits.PEN);                      // Wait for Stop sequence end

   I2C_state_machine = 0;                       // Initialize state machine counter
} // close InitMax6959


// Sequential LEDs update + buttons scan
void UpdateReadMax6959_Seq (void)
{
   switch (I2C_state_machine)
   {
              case 1:
                        temp = SEGnum;
                        if ((SEGnum<0)||(SEGnum>9999))
                                  temp = 0;                 // Check Segnum margins
                        I2CCONbits.SEN = 1;                 // Start I2C frame
              break;

              case 2:
                        if (I2CCONbits.SEN)
                        {I2C_state_machine--; break;}       // Wait for Start sequence END
                   I2CTRN = (0b01110010);                   // Send DEVICE CODE + WRITE
              break;

              case 3:
                        IdleI2C_m                           // If not idle - break (macros)
                   I2CTRN = (0x20);                         // Choose Digit 4 Reg
                                                            // Calculate 10000
              break;

              case 4:
                        IdleI2C_m
                   I2CTRN = (temp/1000);                    // Send  Digit 4       (Reg 0x20)
              break;

              case 5:
                        IdleI2C_m
                   I2CTRN = ((temp/100)%10);                // Send  Digit 3       (Reg 0x21)
              break;

              case 6:
                        IdleI2C_m
                   I2CTRN = ((temp/10)%10);                 // Send  Digit 2       (Reg 0x22)
              break;

              case 7:
                        IdleI2C_m
                   I2CTRN = (temp%10);                      // Send  Digit 1       (Reg 0x23)
              break;

              case 8:
                        IdleI2C_m
                   I2CTRN = (SEGled.leds);                  // Decimal + LEDs   (Reg 0x24)
                        break;
              case 9:
                        IdleI2C_m
                        I2CCONbits.PEN = 1;                 // Stop I2C frame
              break;

              case 10:
                        if(I2CCONbits.PEN)
                          {I2C_state_machine--; break;}     // Wait for Stop sequence end
              break;

              case 11:
                        I2CCONbits.SEN = 1;                 // Start I2C frame
              break;

              case 12:
                        if (I2CCONbits.SEN)
                         {I2C_state_machine--; break;};     // Wait for Start sequence end
```

```
                I2CTRN = (0b01110010);          // Send DEVICE CODE + WRITE
        break;

        case 13:
                IdleI2C_m
           I2CTRN = (0x08);                     // Choose Debounce Reg
        break;

        case 14:
                IdleI2C_m
                I2CCONbits.PEN = 1;             // Stop I2C frame
        break;

        case 15:
                if(I2CCONbits.PEN)
                  {I2C_state_machine--; break;} // Wait for Stop sequence end
                I2CCONbits.SEN = 1;             // Start I2C frame
        break;

        case 16:
                if(I2CCONbits.SEN)
                  {I2C_state_machine--; break;} // Wait for Start sequence end
           I2CTRN = (0b01110011);               // Send DEVICE CODE + WRITE
        break;

        case 17:
                IdleI2C_m
                I2CCONbits.RCEN = 1;
        break;

        case 18:
                if (I2CCONbits.RCEN)
                  {I2C_state_machine--; break;}
        break;

        case 19:
                I2CSTATbits.I2COV = 0;
                if ((I2CRCV==SEGbut.buttons)||(I2CRCV==0)) break;
                SEGbut.buttons = I2CRCV;        // Read buttons       (Reg 0x08)
        break;

        case 20:
                I2CCONbits.ACKDT = 1;           // Not ACK
                I2CCONbits.ACKEN = 1;
                IdleI2C_m
        break;

        case 21:
                I2CCONbits.PEN = 1;             // Stop I2C frame
        break;

        case 22:
                if (I2CCONbits.PEN)             // Wait for Stop sequence end
                  {I2C_state_machine--; break;}
        break;

        case 23:
                I2C_state_machine = 0;          // Reset state machine
                ISR_state_machine++;            // Switch to next task
        break;

        default:                                // Default for "switch (state_machine)"
        break;

  } // close "switch (state_machine)" statement
I2C_state_machine++;                            // Advance internal state machine count
} // close UpdateReadMax6959


// One-shot LEDs + Digits update
void UpdateMax6959_Cont (void)
{
   char temp;
   if ((SEGnum<0)||(SEGnum>9999))
        SEGnum = 0;                             // Number out of range 0..9999

   StartI2C();                                  // Start I2C frame
   while(I2CCONbits.SEN);                       // Wait for Start sequence end

   MasterWriteI2C(0b01110010);                  // Send DEVICE CODE + WRITE
```

```
    IdleI2C();

    MasterWriteI2C(0x20);                   // Choose Digit 4 Reg
    temp = SEGnum/1000;                     // Calculate 10000
    IdleI2C();
    MasterWriteI2C(temp);                   // Send  Digit 4       (Reg 0x20)
    temp = (SEGnum/100)%10;                 // Calculate 1000
    IdleI2C();
    MasterWriteI2C(temp);                   // Send  Digit 3       (Reg 0x21)
    temp = (SEGnum/10)%10;                  // Calculate 100
    IdleI2C();
    MasterWriteI2C(temp);                   // Send  Digit 2       (Reg 0x22)
    temp = SEGnum%10;                       // Calculate 10
    IdleI2C();
    MasterWriteI2C(temp);                   // Send  Digit 1       (Reg 0x23)
    IdleI2C();

    MasterWriteI2C(SEGled.leds);            // Decimal + LEDs   (Reg 0x24)
    IdleI2C();

     StopI2C();                             // Stop I2C frame
     while(I2CCONbits.PEN);                 // Wait for Stop sequence end
} // close UpdateMax6959_Cont


// One-shot buttons scan
void ReadMax6959_Cont (void)
{
    StartI2C();                             // Start I2C frame
    while(I2CCONbits.SEN);                  // Wait for Start sequence end

    MasterWriteI2C(0b01110010);             // Send DEVICE CODE + WRITE
    IdleI2C();
    MasterWriteI2C(0x08);                   // Choose Debounce Reg
    IdleI2C();

     StopI2C();                             // Stop I2C frame
     while(I2CCONbits.PEN);                 // Wait for Stop sequence end
    StartI2C();                             // Start I2C frame
    while(I2CCONbits.SEN);                  // Wait for Start sequence end

    MasterWriteI2C(0b01110011);             // Send DEVICE CODE + WRITE
    IdleI2C();

    SEGbut.buttons = MasterReadI2C();       // Read buttons       (Reg 0x08)
    NotAckI2C();
    IdleI2C();

     StopI2C();                             // Stop I2C frame
     while(I2CCONbits.PEN);                   // Wait for Stop sequence end
} // close ReadMax6959_Cont

// Close I2C session
void CloseI2C(void ) {
  CloseI2C();
} // close CloseI2C

//-----------------------------------
```

Pwm.h

```
// Definitions
//------------------------------
#define FCY             30000000        // Current MCU speed/MIPS
#define FPWM            16000           // Required PWM frequency
//------------------------------

// Functions
//------------------------------
void Init_PWM (void);               // Initialize PWM generator
//------------------------------
```

Pwm.c

```
// Include headers
//------------------------------
#include "p30f6010A.h"
#include "general.h"
#include "pwm.h"
//------------------------------

void Init_PWM(void)
{
                                    // PWMs - output
   TRISEbits.TRISE0 =  0;
   TRISEbits.TRISE1 =  0;
   TRISEbits.TRISE2 =  0;
   TRISEbits.TRISE3 =  0;
   TRISEbits.TRISE4 =  0;
   TRISEbits.TRISE5 =  0;
                                    // PWM control
   TRISEbits.TRISE8 =  1;           // FAULT input
   TRISDbits.TRISD12 = 0;           // BRAKE Fire Output
   TRISDbits.TRISD11 = 0;           // ENABLE Fire Output

PTPER = FCY/FPWM - 1;               // FCY/FPWM - 1 (16kHz)
PWMCON1 = 0x0700;                   // disable PWMs
OVDCON = 0x0000;                    // allow control using OVD
PDC1 = 10;                          // init PWM 1, 2 and 3 to 100
PDC2 = 10;
PDC3 = 10;
// SEVTCMP = PTPER;                 // Set special event compare reg
PWMCON2 = 0b0000000000000110;       // 1:1 postscale values
PTCON = 0x8000;                     // start PWM

FIRE_ENA = 1;                       // Disable PWM driver output
PWMCON1 = 0x0777;                   // Enable PWM outputs
} // Init_PWM
```

```
// Variables, accesible from outside
//-----------------------------------
static const char Scr1[] = {        // Screen 1 Static ROM image

0xFF,0x80,0xBE,0x8A,0x9A,0xAC,0x80,0xBE,0xAA,0xA2,0x80,0xBC,0x8A,0xBC,0x80,0xBE,
0xA0,0xA0,0x80,0x88,0x88,0x80,0x82,0xBE,0x82,0x80,0xBE,0x80,0xBC,0x82,0x84,0x82,
0xBC,0x80,0xBE,0xAA,0xA2,0x80,0x80,0x80,0xBC,0x82,0x84,0x82,0xBC,0x80,0x9C,0xA2,
0xA2,0x9C,0x80,0x82,0xBE,0x82,0x80,0x9C,0xA2,0xA2,0x9C,0x80,0xBE,0x8A,0x9A,0xAC,
0x80,0x80,0x80,0x9C,0xA2,0xA2,0x80,0x9C,0xA2,0xA2,0x9C,0x80,0xBE,0x84,0x88,0xBE,
0x80,0x82,0xBE,0x82,0x80,0xBE,0x8A,0x9A,0xAC,0x80,0x9C,0xA2,0xA2,0x9C,0x80,0x3E,
0xA0,0xA0,0x80,0x80,0x00,0xA4,0xAA,0x92,0x80,0x86,0xB8,0x86,0x80,0xA4,0xAA,0x92,
0x80,0x82,0xBE,0x82,0x80,0xBE,0xAA,0xA2,0x80,0xBC,0x82,0x04,0x82,0xBC,0x80,0xBF,
0x80,0x8E,0x90,0x0E,0x80,0x38,0x88,0x20,0x80,0x80,0xA6,0x92,0x88,0xA4,0x92,0xA0,
0x90,0xA0,0x80,0x80,0xFF,0x10,0x10,0x10,0x28,0x44,0xC4,0x44,0x28,0x10,0x10,0x10,
0xF0,0x10,0x10,0x6C,0x38,0x10,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0x00,0xD0,0xD0,0x10,0xF0,0x00,
0x00,0x00,0x00,0x00,0xF0,0x10,0x10,0x00,0x3C,0x20,0x20,0x00,0x70,0x50,0x20,0x00,
0x40,0x0C,0x04,0x48,0x20,0x10,0x48,0x24,0x40,0x20,0x40,0x00,0x00,0xFE,0x02,0x00,
0xAA,0x00,0x00,0xAA,0x00,0x00,0xFF,0x00,0x1C,0x20,0x3C,0x00,0x60,0x10,0x20,0x10,
0x60,0x00,0x40,0x00,0x1C,0x20,0x1C,0x00,0x00,0xFE,0x02,0x00,0xAA,0x00,0x00,0xAA,
0x00,0xAA,0x00,0x00,0x00,0x03,0x02,0x02,0x00,0x04,0x9E,0x80,0x9E,0x12,0x9E,0x00,
0x1E,0x12,0x9E,0x00,0xFF,0x00,0x00,0x00,0x80,0xC0,0x7F,0xC0,0x81,0x01,0x00,0x00,
0xFF,0x10,0x10,0x6C,0x38,0x10,0x81,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,
0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x81,0x00,0xD1,0x10,0x38,0xFF,0x10,
0x7C,0x82,0x39,0x0D,0x19,0x0D,0x39,0x82,0x7C,0x00,0x03,0x01,0x01,0x01,0x01,0x01,
0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0xFF,0x00,0x00,
0xAA,0x80,0x80,0xAA,0x00,0x00,0xFF,0x00,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,
0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x7F,0x00,0x00,0xAA,0x80,0x80,0xAA,
0x00,0xAA,0x00,0x00,0x00,0xC0,0x80,0x80,0x80,0x00,0xC7,0x42,0xC0,0x00,0x43,0x04,
0x83,0x44,0x03,0x00,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,
0x1F,0x10,0x10,0x6C,0x38,0x10,0x81,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,
0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x81,0x00,0x91,0xD0,0x10,0x1F,0x00,
0x00,0x00,0x01,0x01,0x1F,0x11,0x11,0x00,0x1C,0x20,0x1C,0x00,0x60,0x10,0x20,0x10,
0x60,0x00,0x40,0x00,0x4C,0x24,0x10,0x48,0x24,0x40,0x20,0x40,0x00,0xFC,0x04,0x00,
0xAA,0x00,0x00,0xAA,0x00,0x01,0xFF,0x01,0x24,0x3C,0x24,0x00,0x60,0x10,0x20,0x10,
0x60,0x00,0x40,0x00,0x38,0x14,0x38,0x00,0x00,0xFE,0x02,0x00,0xAA,0x00,0x00,0xAA,
0x00,0xAA,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xE3,0xA2,0x43,0x00,0xE2,0x01,
0xC0,0x02,0xE0,0x00,0xFF,0x00,0x7F,0x01,0x01,0x01,0x01,0x01,0x03,0x10,0x38,0x6C,
0x10,0x10,0x10,0x10,0x10,0x10,0x11,0x11,0x11,0x11,0x11,0x11,0x01,0x3D,0x15,0x01,
0x3D,0x31,0x01,0x11,0x11,0x11,0x11,0x11,0x11,0x11,0x10,0x11,0x11,0x10,0x10,0x10,
0x10,0x10,0x10,0x10,0x10,0x10,0x03,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,
0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x7F,0x00,0x00,
0xAA,0x80,0x80,0xAA,0x00,0x00,0xFF,0x00,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,
0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x7F,0x00,0x00,0xAA,0x80,0x80,0xAA,
0x00,0xAA,0x00,0x00,0x00,0xC0,0x40,0x40,0x00,0x10,0x79,0x00,0x79,0x48,0x78,0x01,
0x78,0x49,0x78,0x00,0xFF,0x00,0x01,0x48,0x55,0x24,0x01,0x0C,0x71,0x0C,0x01,0x48,
0x55,0x24,0x01,0x40,0x01,0x00,0x01,0x48,0x55,0x24,0x01,0x04,0x7D,0x04,0x01,0x78,
0x15,0x78,0x01,0x04,0x7D,0x04,0x01,0x3C,0x41,0x7C,0x01,0x48,0x55,0x24,0x01,0x00,
0x01,0x10,0x31,0x70,0x31,0x10,0x01,0x00,0x01,0x54,0x01,0x00,0x01,0x00,0x01,0x00,
0x7D,0x14,0x09,0x00,0x01,0x00,0x01,0x00,0x01,0x00,0x01,0x00,0x49,0x54,0x25,0x00,
0x00,0x00,0x00,0x00,0x00,0x01,0xFF,0x01,0x04,0x3C,0x04,0x00,0x60,0x10,0x20,0x10,
0x60,0x00,0x40,0x04,0x00,0x18,0x24,0x00,0x00,0xFE,0x02,0x00,0xAA,0x00,0x00,0xAA,
0x01,0x01,0x01,0x00,0x01,0x00,0x01,0x00,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,
0x01,0x01,0x01,0x01,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x55,0x00,0x00,0x00,0x00,0xFF,0x01,
0x01,0x01,0x01,0x01,0x01,0xFF,0x00,0x00,0x00,0x00,0xFF,0x01,0x01,0x01,0x01,0x01,
0x01,0xFF,0x00,0x00,0x00,0x00,0xFF,0x00,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,
0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x7F,0x00,0x00,0xAA,0x80,0x80,0xAA,
0xFF,0x01,0x49,0x55,0x25,0x01,0x7D,0x45,0x39,0x01,0x01,0x7D,0x45,0x7D,0x01,0x7D,
0x55,0x7D,0x01,0x75,0x55,0x5D,0x01,0x75,0x55,0x5D,0x01,0x01,0x11,0x11,0x11,0x01,
0x01,0x39,0x45,0x45,0x01,0x39,0x45,0x45,0x39,0x01,0x45,0x29,0x11,0x29,0x45,0x01,
0x81,0x41,0x01,0x01,0x7D,0x11,0x29,0x45,0x01,0x39,0x45,0x45,0x39,0x01,0x45,0x65,
0x55,0x4D,0x01,0x7D,0x01,0x05,0x09,0x71,0x09,0x05,0x01,0x81,0x41,0x00,0x01,0x7D,
0x41,0x41,0x41,0x01,0x39,0x45,0x44,0x39,0x01,0x04,0x09,0x71,0x09,0x05,0x01,0x01,
0x11,0x11,0x10,0x01,0x01,0x75,0x55,0x5D,0x01,0x7D,0x45,0x7D,0x01,0x7D,0x45,0x7D,
0x01,0x5D,0x55,0x7D,0x01,0x01,0x39,0x45,0x01,0x39,0x29,0x00,0x44,0x38,0x00,0xFE
};
```

```
// Definitions
//------------------------------
#define VBAR_1                  7           // Vertical bar 1 location in psed4x8
//------------------------------

// Variables, accesible from outside
//----------------------------------
static const char font4x8[] = {             // Numbers + signs, 4x8

0x10,0x38,0x10,0x00,                        // +
0x40,0x20,0x00,0x00,                        // ,
0x10,0x10,0x10,0x00,                        // -
0x60,0x60,0x00,0x00,                        // .
0x20,0x10,0x08,0x00,                        // /

0x7C,0x44,0x7C,0x00,                        // 0
0x48,0x7C,0x40,0x00,                        // 1
0x74,0x54,0x5C,0x00,                        // 2
0x54,0x54,0x7C,0x00,                        // 3
0x1C,0x10,0x78,0x00,                        // 4
0x5C,0x54,0x74,0x00,                        // 5
0x7C,0x54,0x74,0x00,                        // 6
0x04,0x04,0x7C,0x00,                        // 7
0x7C,0x54,0x7C,0x00,                        // 8
0x5C,0x54,0x7C,0x00,                        // 9

0x00,0x28,0x00,0x00,                        // :
0x40,0x28,0x00,0x00,                        // ;
0x10,0x28,0x44,0x00,                        // <
0x28,0x28,0x28,0x00,                        // =
0x44,0x28,0x10,0x00,                        // >
0x04,0x54,0x0C,0x00             // ?
};

static const char psed4x8[] = {   // Pseudo graphics, 4x8

0xFF,0x80,0x80,0x80,                        // |_
0x80,0x80,0x80,0xFF,                        // _|
0xC0,0xC0,0xC0,0xC0,                        // 1x_
0xF0,0xF0,0xF0,0xF0,                        // 2x_
0xFC,0xFC,0xFC,0xFC,                        // 3x_
0xF0,0xF0,0xF0,0xF0,                        // 4x_

0xFF,0x01,0x01,0xFF,                        // |-|
0xFF,0x00,0x00,0xFF,                        // | |
0xFF,0xC0,0xC0,0xFF,                        // |_|
0xFF,0xF0,0xF0,0xFF,                        // |=|
0xFF,0xFC,0xFC,0xFF,                        // |=|x2
0xFF,0xFF,0xFF,0xFF,                        // |=|x4

0x00,0xFF,0x00,0x00,                        // | |
0xC0,0xFF,0xC0,0xC0,                        // | down
0x30,0xFF,0x30,0x30,                        // |- middle
0x0C,0xFF,0x0C,0x0C,                        // |- middle x2
0x03,0xFF,0x03,0x03,                        // |-| up

0x40,0x40,0x40,0x7C,                        // Flag OFF
0x7C,0x7C,0x7C,0x7C                         // Flag ON
};

static const char font5x8[] = {             // ASCII, 5x8

0x00, 0x00, 0x00, 0x00, 0x00, 0x00,         // space
0x00, 0x00, 0x5F, 0x00, 0x00, 0x00,         // !
0x00, 0x07, 0x00, 0x07, 0x00, 0x00,         // "
0x14, 0x7F, 0x14, 0x7F, 0x14, 0x00,         // #
0x24, 0x2A, 0x7F, 0x2A, 0x12, 0x00,         // $
0x23, 0x13, 0x08, 0x64, 0x62, 0x00,         // %
0x36, 0x49, 0x55, 0x22, 0x50, 0x00,         // &
0x00, 0x05, 0x03, 0x00, 0x00, 0x00,         // '
0x00, 0x1C, 0x22, 0x41, 0x00, 0x00,         // (
0x00, 0x41, 0x22, 0x1C, 0x00, 0x00,         // )
0x08, 0x2A, 0x1C, 0x2A, 0x08, 0x00,         // *
0x08, 0x08, 0x3E, 0x08, 0x08, 0x00,         // +
0x00, 0x50, 0x30, 0x00, 0x00, 0x00,         // ,
0x08, 0x08, 0x08, 0x08, 0x08, 0x00,         // -
0x00, 0x60, 0x60, 0x00, 0x00, 0x00,         // .
```

```
0x20, 0x10, 0x08, 0x04, 0x02, 0x00,     // /
0x3E, 0x51, 0x49, 0x45, 0x3E, 0x00,     // 0
0x00, 0x42, 0x7F, 0x40, 0x00, 0x00,     // 1
0x42, 0x61, 0x51, 0x49, 0x46, 0x00,     // 2
0x21, 0x41, 0x45, 0x4B, 0x31, 0x00,     // 3
0x18, 0x14, 0x12, 0x7F, 0x10, 0x00,     // 4
0x27, 0x45, 0x45, 0x45, 0x39, 0x00,     // 5
0x3C, 0x4A, 0x49, 0x49, 0x30, 0x00,     // 6
0x01, 0x71, 0x09, 0x05, 0x03, 0x00,     // 7
0x36, 0x49, 0x49, 0x49, 0x36, 0x00,     // 8
0x06, 0x49, 0x49, 0x29, 0x1E, 0x00,     // 9
0x00, 0x36, 0x36, 0x00, 0x00, 0x00,     // :
0x00, 0x56, 0x36, 0x00, 0x00, 0x00,     // ;
0x00, 0x08, 0x14, 0x22, 0x41, 0x00,     // <
0x14, 0x14, 0x14, 0x14, 0x14, 0x00,     // =
0x41, 0x22, 0x14, 0x08, 0x00, 0x00,     // >
0x02, 0x01, 0x51, 0x09, 0x06, 0x00,     // ?
0x32, 0x49, 0x79, 0x41, 0x3E, 0x00,     // @
0x7E, 0x11, 0x11, 0x11, 0x7E, 0x00,     // A
0x7F, 0x49, 0x49, 0x49, 0x36, 0x00,     // B
0x3E, 0x41, 0x41, 0x41, 0x22, 0x00,     // C
0x7F, 0x41, 0x41, 0x22, 0x1C, 0x00,     // D
0x7F, 0x49, 0x49, 0x49, 0x41, 0x00,     // E
0x7F, 0x09, 0x09, 0x01, 0x01, 0x00,     // F
0x3E, 0x41, 0x41, 0x51, 0x32, 0x00,     // G
0x7F, 0x08, 0x08, 0x08, 0x7F, 0x00,     // H
0x00, 0x41, 0x7F, 0x41, 0x00, 0x00,     // I
0x20, 0x40, 0x41, 0x3F, 0x01, 0x00,     // J
0x7F, 0x08, 0x14, 0x22, 0x41, 0x00,     // K
0x7F, 0x40, 0x40, 0x40, 0x40, 0x00,     // L
0x7F, 0x02, 0x04, 0x02, 0x7F, 0x00,     // M
0x7F, 0x04, 0x08, 0x10, 0x7F, 0x00,     // N
0x3E, 0x41, 0x41, 0x41, 0x3E, 0x00      ,// O
0x7F, 0x09, 0x09, 0x09, 0x06, 0x00,     // P
0x3E, 0x41, 0x51, 0x21, 0x5E, 0x00,     // Q
0x7F, 0x09, 0x19, 0x29, 0x46, 0x00,     // R
0x46, 0x49, 0x49, 0x49, 0x31, 0x00,     // S
0x01, 0x01, 0x7F, 0x01, 0x01, 0x00,     // T
0x3F, 0x40, 0x40, 0x40, 0x3F, 0x00,     // U
0x1F, 0x20, 0x40, 0x20, 0x1F, 0x00,     // V
0x7F, 0x20, 0x18, 0x20, 0x7F, 0x00,     // W
0x63, 0x14, 0x08, 0x14, 0x63, 0x00,     // X
0x03, 0x04, 0x78, 0x04, 0x03, 0x00,     // Y
0x61, 0x51, 0x49, 0x45, 0x43, 0x00,     // Z
0x00, 0x00, 0x7F, 0x41, 0x41, 0x00,     // [
0x02, 0x04, 0x08, 0x10, 0x20, 0x00,     // "\"
0x41, 0x41, 0x7F, 0x00, 0x00, 0x00,     // ]
0x04, 0x02, 0x01, 0x02, 0x04, 0x00,     // ^
0x40, 0x40, 0x40, 0x40, 0x40, 0x00,     // _
0x00, 0x01, 0x02, 0x04, 0x00, 0x00,     // `
0x20, 0x54, 0x54, 0x54, 0x78, 0x00,     // a
0x7F, 0x48, 0x44, 0x44, 0x38, 0x00,     // b
0x38, 0x44, 0x44, 0x44, 0x20, 0x00,     // c
0x38, 0x44, 0x44, 0x48, 0x7F, 0x00,     // d
0x38, 0x54, 0x54, 0x54, 0x18, 0x00,     // e
0x08, 0x7E, 0x09, 0x01, 0x02, 0x00,     // f
0x08, 0x14, 0x54, 0x54, 0x3C, 0x00,     // g
0x7F, 0x08, 0x04, 0x04, 0x78, 0x00,     // h
0x00, 0x44, 0x7D, 0x40, 0x00, 0x00,     // i
0x20, 0x40, 0x44, 0x3D, 0x00, 0x00,     // j
0x00, 0x7F, 0x10, 0x28, 0x44, 0x00,     // k
0x00, 0x41, 0x7F, 0x40, 0x00, 0x00,     // l
0x7C, 0x04, 0x18, 0x04, 0x78, 0x00,     // m
0x7C, 0x08, 0x04, 0x04, 0x78, 0x00,     // n
0x38, 0x44, 0x44, 0x44, 0x38, 0x00,     // o
0x7C, 0x14, 0x14, 0x14, 0x08, 0x00,     // p
0x08, 0x14, 0x14, 0x18, 0x7C, 0x00,     // q
0x7C, 0x08, 0x04, 0x04, 0x08, 0x00,     // r
0x48, 0x54, 0x54, 0x54, 0x20, 0x00,     // s
0x04, 0x3F, 0x44, 0x40, 0x20, 0x00,     // t
0x3C, 0x40, 0x40, 0x20, 0x7C, 0x00,     // u
0x1C, 0x20, 0x40, 0x20, 0x1C, 0x00,     // v
0x3C, 0x40, 0x30, 0x40, 0x3C, 0x00,     // w
0x44, 0x28, 0x10, 0x28, 0x44, 0x00,     // x
0x0C, 0x50, 0x50, 0x50, 0x3C, 0x00,     // y
0x44, 0x64, 0x54, 0x4C, 0x44, 0x00,     // z
0x00, 0x08, 0x36, 0x41, 0x00, 0x00,     // {
0x00, 0x00, 0x7F, 0x00, 0x00, 0x00,     // |
0x00, 0x41, 0x36, 0x08, 0x00, 0x00,     // }
0x08, 0x08, 0x2A, 0x1C, 0x08, 0x00,     // ->
```

```
0x08, 0x1C, 0x2A, 0x08, 0x08, 0x00,     // <-
0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0x00,     // Bar 4x4
0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00      // Bar 4x8
};
```